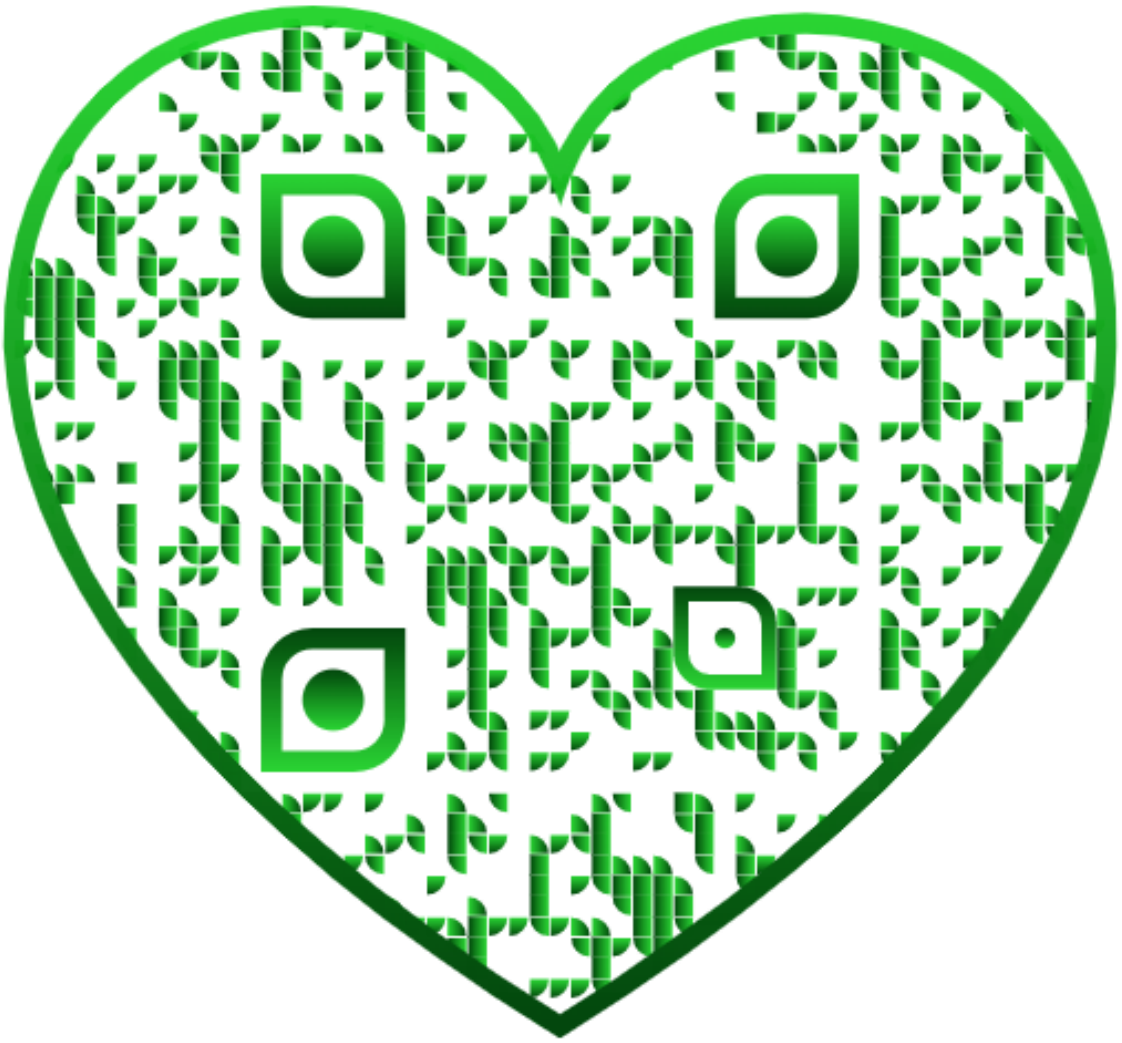
Créer son jeu.fr
En partant de zéro, sans connaissances

# Apprendre le Game Maker Language pour créer ses jeux comme un pro

Maxime G.



Copyright © creersonjeu.fr

# PRÉFACE

Si vous êtes en train de lire ce livre, c'est que vous faites partie de ces personnes qui rêvent de créer leur propre jeu vidéo. Rassurez-vous, vous n'êtes pas tout seul et c'est pour cela que j'ai fondé Creersonjeu.fr !

Je transmets depuis maintenant près de deux ans mes connaissances dans la création de jeux vidéo, à travers mon blog et ma chaîne YouTube.

Je pense qu'il est tout à fait possible de créer le jeu de ses rêves, si l'on en a la volonté et un minimum de connaissances.

Je tenais à vous remercier d'être toujours plus à me suivre, c'est grâce à vous que ce livre voit le jour ! J'espère sincèrement qu'il vous plaira et vous aidera à progresser. Si vous ne faites pas encore partie de la communauté Creersonjeu.fr, vous pouvez nous rejoindre via les liens suivants :

***YouTube***

***Twitter***

***Facebook***

Ou me contacter via mon email : ***contact@creersonjeu.fr***

## LICENCE À RESPECTER

Cette oeuvre est protégée par le droit d'auteur et strictement réservée à l'usage privé du client. Toute reproduction ou diffusion au profit de tiers, à titre gratuit ou onéreux, de tout ou partie de cette oeuvre est strictement interdite et constitue une contrefaçon prévue par les articles L 335-2 et suivants du Code de la propriété intellectuelle.

# UN PETIT MOT POUR COMMENCER

Bonjour les créateurs et bienvenue dans mon premier livre sur Game Maker Studio. Je suis Maxime de creersonjeu.fr !

Ce livre s'adresse principalement aux débutants et a pour but de vous enseigner les bases du *Game Maker Language* (par soucis de clareté, il pourra être abrégé en *GML* dans le texte), ainsi que certains concepts plus avancés afin de vous permettre de créer vos propres jeux vidéo. C'est un langage très puissant, qui vous permettra une fois maitrisé de réaliser dans vos jeux tout ce qui vous passe par la tête ! Grâce à lui, vous deviendrez enfin maître de votre création ☺

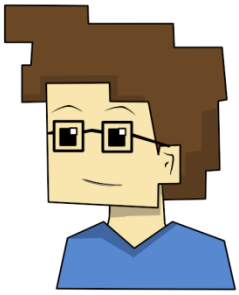
Je suis sûr que la plupart d'entre vous ont déjà téléchargé mon Ebook gratuit « *Créer son premier jeu vidéo : par où commencer ?* » (si ce n'est pas déjà fait je vous encourage à le faire). L'Ebook que vous avez sous les yeux est beaucoup plus complet afin d'être sûr de retranscrire dans les meilleurs conditions possibles mes connaissances sur le Game Maker Language. Certains concepts ont été simplifiés afin que tout le monde puisse comprendre facilement les sujets abordés et commencer à créer son propre jeu avec des premières bases solides.

Veuillez noter que ce guide se base sur l'utilisation de *Game Maker Studio 2*. Bien que de nombreuses fonctions soient similaires avec Game Maker Studio premier du nom, il se peut que certaines ne fonctionnent pas sur ce dernier.

J'espère sincèrement que vous apprendrez de nombreuses choses avec cet ouvrage. N'hésitez pas à me faire parvenir vos remarques et commentaires, je serais ravi d'avoir votre avis et, pourquoi pas, mettre à jour le livre.

Notez également que pour aller encore plus loin, il existe la documentation officielle Game Maker Studio (en anglais).

Bonne lecture !



Copyright © creersonjeu.fr

# SOMMAIRE

## INTRODUCTION

Qu'est-ce que le Game Maker Language ? ..... *2*

Le Drag'n drop et ses limites ..... *4*

## FONCTIONNEMENT DU LANGAGE

Syntaxe, convention de nommage et règles de base ..... *7*

Les opérateurs ..... *10*

Les variables ..... *14*

Les semicolons ..... *19*

Les arrays ..... *20*

Les commentaires ..... *23*

Les conditions ..... *24*

Les boucles ..... *31*

Les fonctions ..... *36*

Copyright © creersonjeu.fr

# LES FONCTIONS ET VARIABLES PRÉINTÉGRÉES DANS GAME MAKER STUDIO 2

Fonctions et variables liées aux room ..... *43*

Fonctions et variables d'affichage ..... *47*

Fonctions et variables de gestion des déplacements, collisions ..... *51*

Fonctions de contrôle de partie ..... *55*

Fonctions audio ..... *57*

Fonctions et variables liées aux instances ..... *59*

Fonctions de dessin ..... *62*

Fonctions de contrôles in-game ..... *72*

Copyright © creersonjeu.fr

# INTRODUCTION

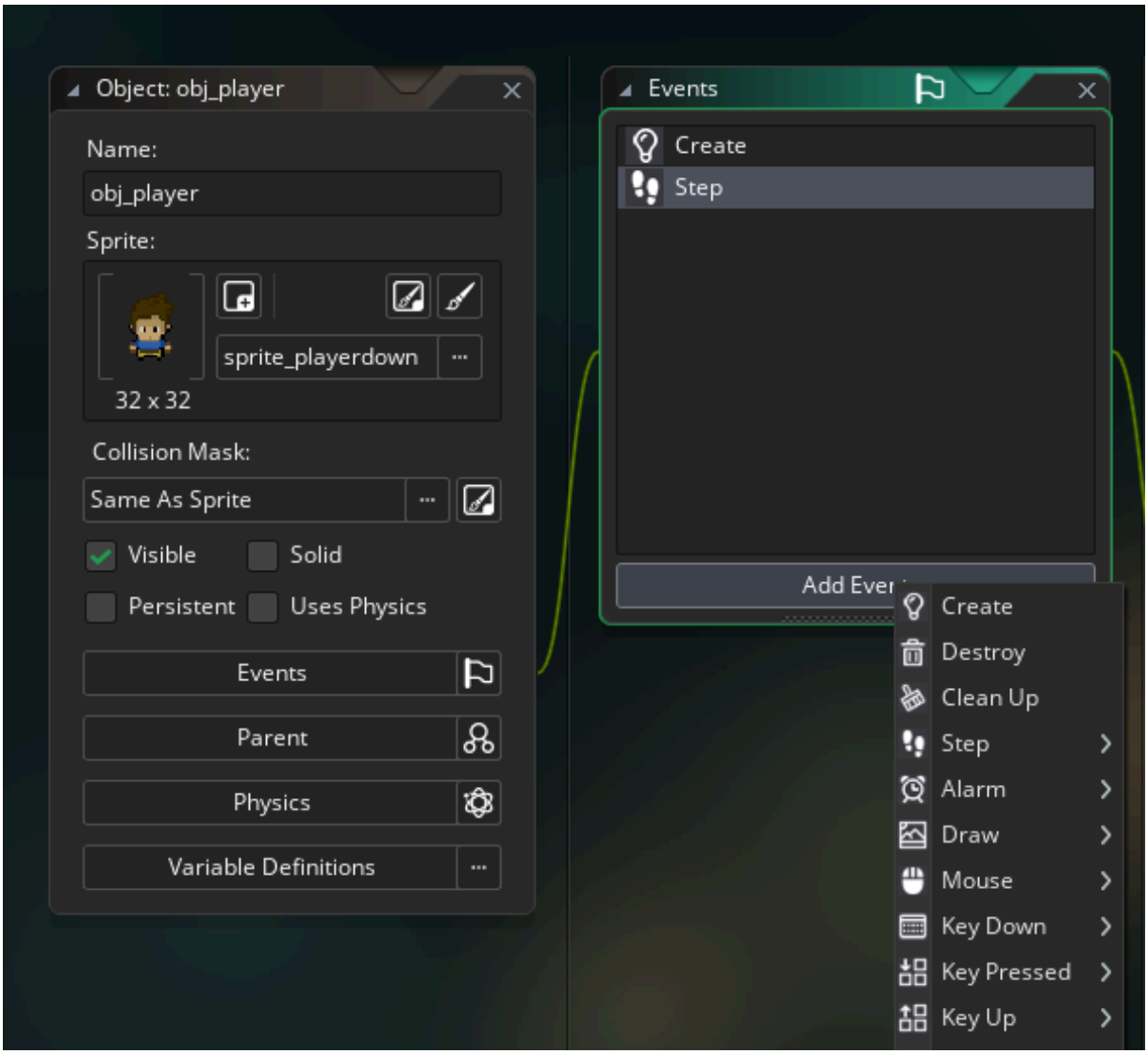


Copyright © creersonjeu.fr

## QU'EST-CE QUE LE GAME MAKER LANGUAGE ?

Le Game Maker Language est le langage de programmation intégré à Game Maker Studio. Comme tout langage de ce type, celui-ci utilise une syntaxe spécifique, des mots clés et a sa propre logique de fonctionnement.

Il permet d'interagir avec l'ensemble des éléments créés via l'interface Game Maker : les objets, les sprites, les sons, les scripts… Ce dernier peut s'écrire soit dans les évènements de vos objets : step event, draw event, alarm…



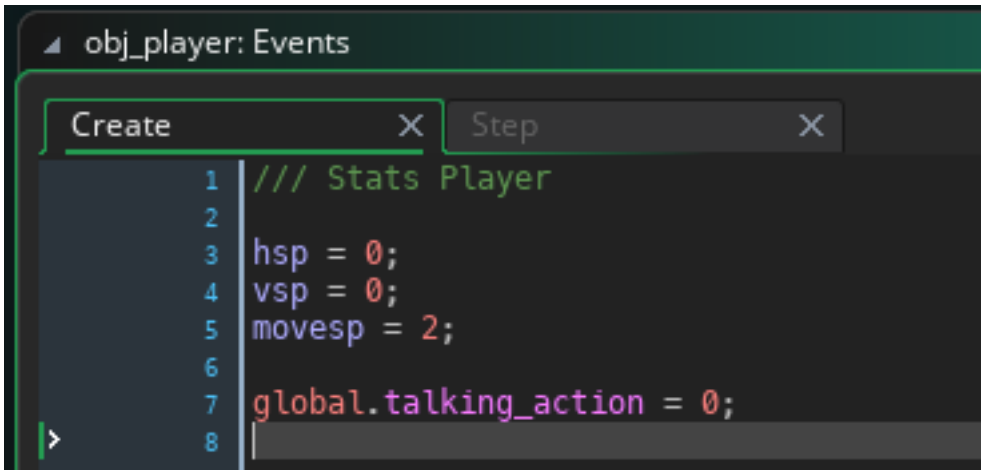

*Petit rappel, les évènements ou « event » en anglais représentent différents niveaux d'interaction d'un objet. Ils permettent de contrôler les actions ainsi que l'état de ce dernier dans le jeu.*



Copyright © creersonjeu.fr

… soit dans les scripts que vous aurez créés spécialement pour votre jeu. Nous verrons ceux-ci plus tard.

Voici un exemple de lignes de Game Maker Language rédigé dans l'interface Game Maker :




Le GML est très puissant, il permet s'il est bien maîtrisé de réaliser à peu près tout et n'importe quoi.

Tout cela c'est très bien, mais je suis sûr que certains d'entre vous l'ont remarqué, il existe une autre manière pour programmer des jeux dans Game Maker : le Drag'n Drop.

Pourquoi je ne vous conseille surtout pas de l'utiliser ? On se retrouve tout de suite au chapitre suivant pour en parler ;)

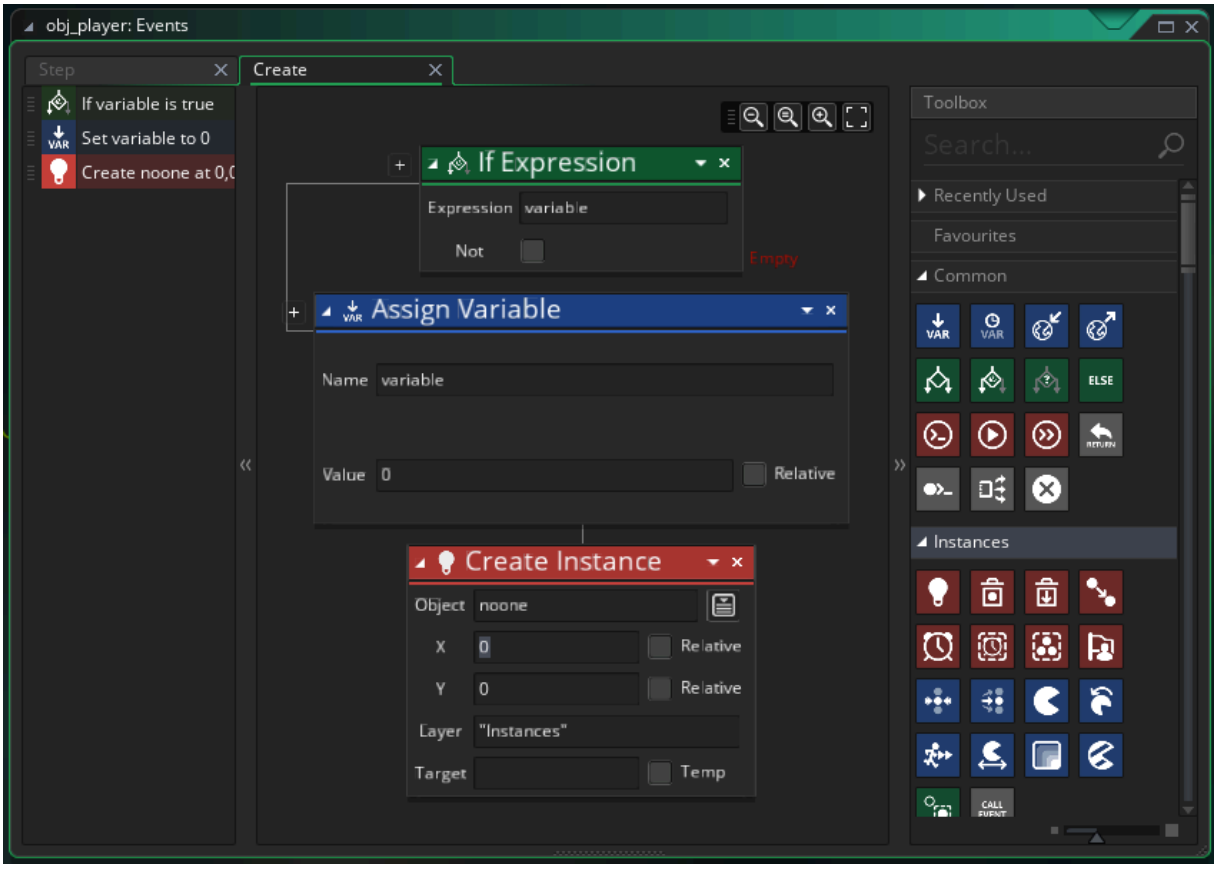


Copyright © creersonjeu.fr

## LE DRAG'N DROP ET SES LIMITES

Tout d'abord le Drag'n Drop, qu'est-ce que c'est ?

C'est un système de *« visual scripting »* intégré à Game Maker Studio et destiné aux débutants. Il permet très simplement et sans connaissances de réaliser les premières interactions de votre jeu. Pour cela il suffit de faire glisser et déposer (d'où le nom Drag'n Drop) des **briques**, correspondant à des bouts de code, qui peuvent interagir entre elles. C'est un système très pratique lorsque l'on débute, car il est très visuel et permet de se concentrer uniquement sur la logique du jeu et non le langage de programmation.




Aussi pratique soit-il, je vous conseille cependant d'abandonner au plus vite la pratique du Drag'n Drop.

Pourquoi me direz-vous ?

En réalité, bien que le Drag'n Drop soit également très puissant, ce n'est qu'une solution secondaire à la programmation. Vous ne manipulerez jamais réellement de code en utilisant



Copyright © creersonjeu.fr

ce système. Cela vous bloquera forcément à un moment donné, et vous limitera dans les choses que vous souhaiterez mettre en place pour votre jeu.

J'en sais quelque chose, j'ai été comme vous ! Je peux vous dire que lorsque l'on passe du côté GML, on ne revient pas en arrière !

Vous pouvez bien entendu l'utiliser lors de la découverte de Game Maker Studio, pour vous familiariser avec le logiciel et sa logique, mais pensez ensuite à utiliser le plus vite possible le Game Maker Language.

*PS : à noter que vous pouvez tout à fait utiliser le Game Maker Language ET le Drag'n Drop en même temps dans un projet. Pratique lorsque l'on ne maîtrise pas encore à 100% le langage !*



Copyright © creersonjeu.fr

# FONCTIONNEMENT DU LANGAGE



Copyright © creersonjeu.fr

# SYNTAXE, CONVENTION DE NOMMAGE ET RÈGLES DE BASE

Une question que l'on m'a souvent posée sur le blog et sur Youtube :

*« Est-il nécessaire de connaître un autre langage de programmation avant de se lancer dans le GML ? »*

La réponse est… Non ! Le Game Maker Language est à mon sens plutôt simple à appréhender une fois que l'on a assimilé les bases. Vous pouvez donc vous lancer à corps perdu dans l'apprentissage de celui-ci, même en étant un parfait débutant. Je dirais même que celui-ci vous aidera par la suite à apprendre de nouveau langages ;)

**a. Comment se construit le GML ?**

Comme tout langage de programmation, le GML est rédigé sous forme de **blocs d'instructions**.

Un **bloc d'instructions** est composé d'une ou plusieurs lignes de code (appelées en anglais «**statements** », ou « **expressions** » en français) qui vont être lues et interprétées par Game Maker Studio. Ensemble, ces lignes permettent d'effectuer des actions dans notre jeu.

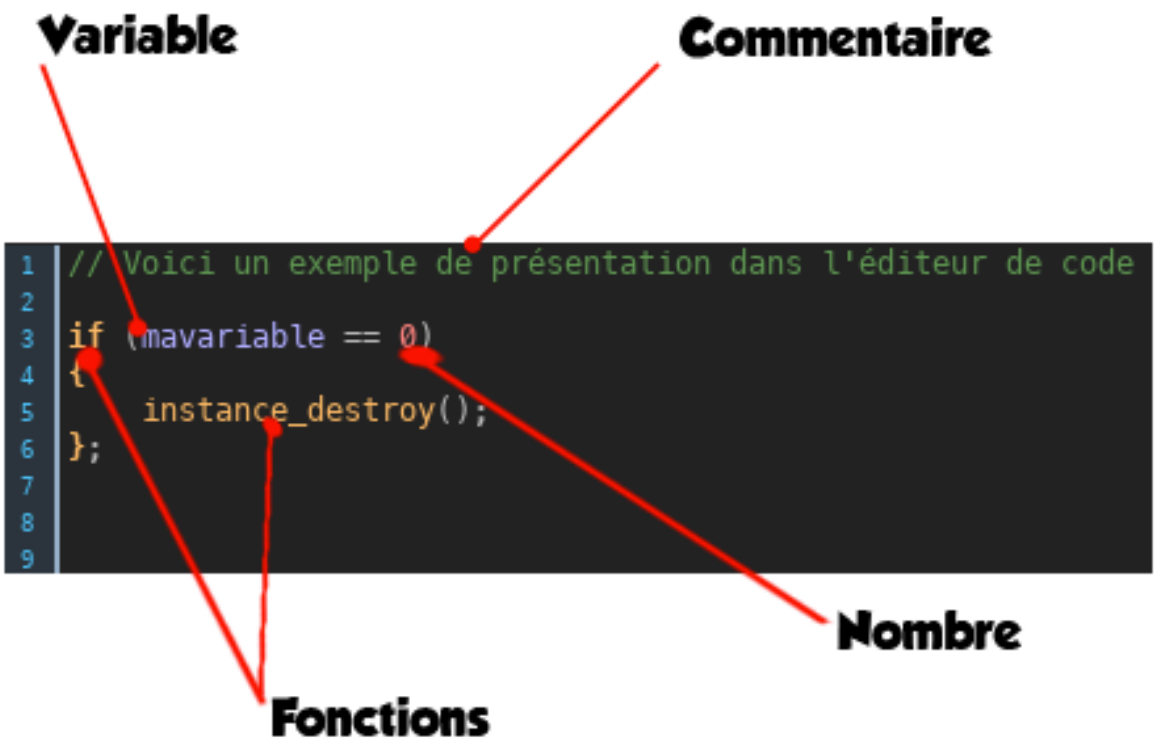
Les blocs d'instructions sont écrits les uns à la suite des autres, peuvent interagir entre eux, et son lus par Game Maker Studio de haut en bas.

**b. Où écrire ces blocs d'instructions ?**

Chose très pratique, Game Maker Studio possède un éditeur de code intégré, qui permet d'afficher les numéros de lignes, différentes couleurs en fonction du type de valeur affichée, des notifications d'erreurs lorsqu'il en détecte… Très pratique je vous dis !



Copyright © creersonjeu.fr




En parlant de couleurs, vous trouverez ci-dessous les différents codes couleurs en fonction des types d'instructions affichés dans l'éditeur (si certains ne vous disent rien ne vous inquiétez pas, on y reviendra par la suite) :

- **Vert** : les commentaires, qui ne sont pas interprétés par Game Maker Studio une fois le jeu lancé (ils servent uniquement à donner des indications aux personnes lisant le code)
- **Violet** : les variables (qui peuvent également être jaune)
- **Orange** : les fonctions
- **Rouge** : les nombres, les booléens (les constantes)

**c. Les statements**

Un statement est une combinaison de plusieurs opérateurs et d'opérandes.

*« Opérateurs… Opérandes… Pardon ? »*

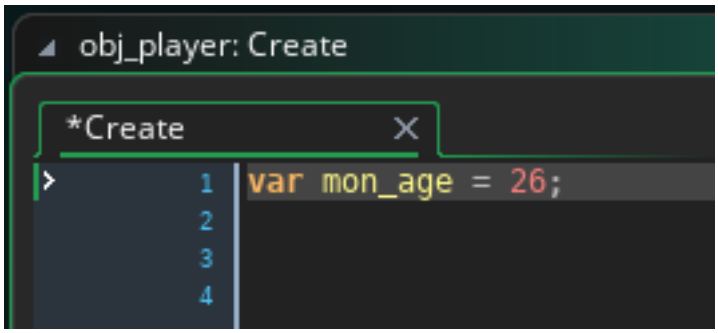
Ne vous inquiétez pas, je vous explique !



Copyright © creersonjeu.fr

Je vous ai dit tout à l'heure que l'écriture d'un programme en Game Maker Language se faisait en rédigeant plusieurs blocs d'instructions à la suite. Ces blocs d'instructions sont composés, entre autres, de statements.

Pour que vous compreniez bien, imaginons que nous voulions déclarer une variable appelée « mon âge ».




Dans cet exemple, nous avons attribué à la variable *mon_age* le valeur 26. Ici *mon_age* et *26* sont appelés des ***opérandes***, et « = » est un ***opérateur*** (plus précisément un opérateur d'assignement, car il permet d'assigner la valeur 26 à la variable identifiée par le nom « mon_age »).

La ligne *var mon_age = 26* ; est donc appelée un statement !



Copyright © creersonjeu.fr

# LES OPÉRATEURS

Nous venons de voir la différence entre un opérateur et un opérande. Cependant, il existe de nombreux types d'opérateurs permettant de réaliser différentes actions au sein de notre code ! Nous allons détailler ces actions ici.

**a. Attribuer une valeur à une variable**

Cette action se réalise grâce à l'opérateur d'assignement « = ».

Exemple : i = 3 ;

**b. Comparer des valeurs**

Cette action se réalise grâce aux opérateurs suivants, et ne retourne uniquement que des résultats booléens. Cela signifie que le résultat de cette comparaison ne peut être que vrai, ou faux (*true* ou *false* en anglais). Il est possible de comparer des nombres bien entendu, mais également des chaînes de caractères (strings).

- « == » permet de déterminer si deux valeurs sont égales

  Exemple :

  *a = 3 ;*

  *b = 3 ;*

  *a == b ; // a est bien égale à b, cela retourne  donc true !*

- « != » permet de déterminer si deux valeurs sont différentes
- « <= » et « >= » permet de déterminer si une valeur est « inférieure ou égale » ou « supérieure ou égale » à une autre valeur
- « < » et « > » permet de déterminer si une valeur est « strictement inférieure » ou « strictement supérieure » à une autre



Copyright © creersonjeu.fr

### c. Combiner plusieurs comparaisons de valeurs

Afin de pouvoir combiner plusieurs comparaisons, il faut utiliser les opérateurs suivant :

- « && » signifie « et » (« and » en anglais)

  Exemple :

```
if (a == 3 && b == 3)
{

};

/* Cela signifie « Si a est égal à 3 (true)
ET b est également égal à 3 (true), alors fait quelque chose.. » */

```

- « || » signifie « ou » (« or » en anglais)

  Exemple :

```
if (a == 3 || b == 3)
{

};
/* Cela signifie « Si a est égal à 3
OU que b est égal à 3, alors fait quelque chose… » */

```

### d. Les opérateurs arithmétiques

Ceux-ci vous les connaissez, ils nous servent à effectuer des opérations dans notre code :

- L'opérateur « + » pour additionner
- L'opérateur « - » pour soustraire
- L'opérateur « * » pour multiplier
- Enfin l'opérateur « / » pour diviser



Copyright © creersonjeu.fr

### e. Les opérateurs d'incrémentation et de décrémentation

Incrémenta…décrémenta quoi ?!

Du calme du calme, ces mots peuvent faire peur mais leur signification est en réalité très simple. En informatique, on appelle **incrémentation** le fait d'ajouter une valeur entière à une autre. La **décrémentation** est l'action de soustraire une valeur entière. Ces opérations sont très utiles et sont possibles grâce aux opérateurs :

- ++ pour incrémenter
- -- pour décrémenter

Exemple :

```
a = 3;

if (a == 3)
{
    a ++ ; //a sera donc égale à 4 !
};
```

### f. L'opérateur de négation

Nous l'avons utilisé brièvement dans les paragraphes précédents. Vous verrez que cet opérateur se révèlera être très utile pour vos programmes. Il existe en effet un opérateur qui vous permet d'ajouter une négation aux interprétations de votre code par Game Maker. Il s'utilise exclusivement avec des opérateurs booléens (*true* et *false*).

Cet opérateur, c'est le point d'exclamation : !

SI vous ne voyez pas exactement à quoi cela peut servir, laissez-moi vous expliquer comme à mon habitude à travers un exemple :

true == true ;



Copyright © creersonjeu.fr

Jusque-là on est d'accord ? Et si maintenant j'ajoute l'opérateur de négation « ! » ?

! true == false ;

Négation !

Prenons un exemple plus concret :

```
a = 3;

if !(a == 3)
{
    a ++ ;
};

/* Cela signifie : Si la variable a n'est PAS EGALE à 3,
alors on incrémente de 1 la variable a */
```



Copyright © creersonjeu.fr

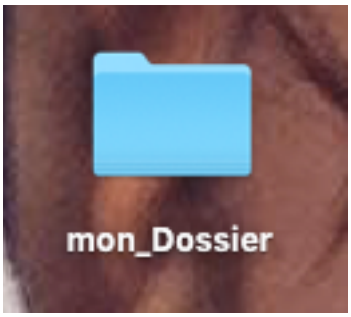
## LES VARIABLES

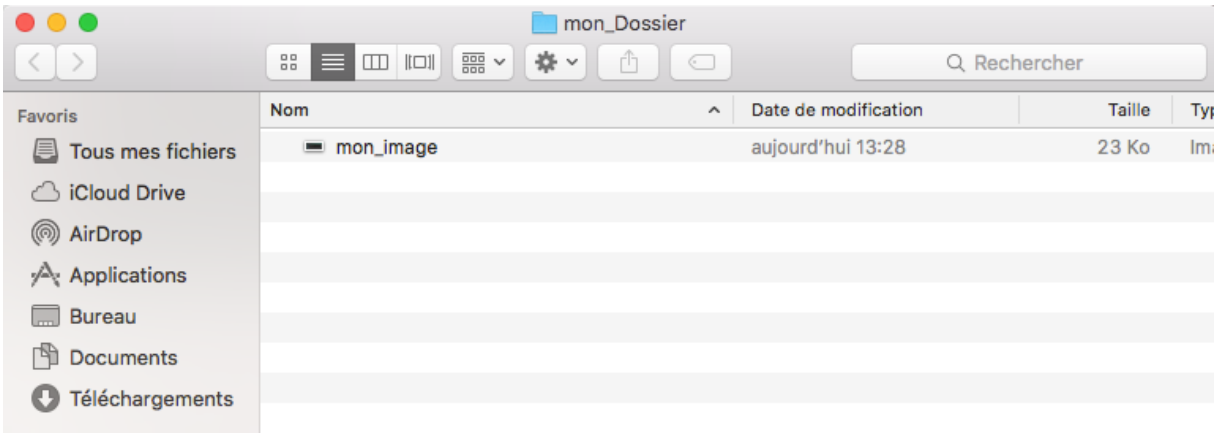
Lorsque l'on apprend à programmer, une des premières notions que l'on apprend se nomme *les variables*.

Mais qu'est-ce qu'une variable au juste ?

Une variable est une **structure de données**. Elle permet d'associer **un nom** avec **une valeur**. Cette valeur peut changer au cours du temps, et peut avoir différents types : chaîne de caractères ("string" en anglais, un ensemble de mots), un nombre, un résultat booléen (un type particulier de valeur qui peut être soit vraie, soit fausse. *True* ou *false* en anglais), et même la valeur d'une autre variable !

Pour que vous compreniez mieux, imaginez un dossier sur le bureau de votre ordinateur. Vous pouvez nommer ce dossier, et y mettre le fichier que vous souhaitez à l'intérieur (une vidéo, une image…).






Copyright © creersonjeu.fr

C'est la même chose avec les variables !

Voyez plutôt :

```
1 var mon_Dossier = "mon_Image";
2
3
```

Comme vous pouvez le voir ici, la variable à bien le même nom que le dossier, et elle contient une chaîne de caractères, une string, s'appelant *mon_Image*). On aurait très bien pu y mettre également un nombre.

Vous remarquerez que l'on a mis qu'une seule valeur dans notre variable. En effet, si nous avions voulu en ajouter d'autres, il aurait fallu utiliser un autre type de variable : un **array** ! Nous y reviendrons par la suite ;)

*" Mais pourquoi dans ton exemple ta variable s'appelle "mon_Dossier" et pas plutôt "mon dossier" tout simplement ? "*

Si vous l'aviez remarqué, bravo ! En effet, bien que vous puissiez nommer comme bon vous semble vos variables, il y a cependant quelques règles à respecter :

- Le nom doit impérativement commencer par une lettre
- Les accents sont interdits (pas de "var évènement" par exemple)
- Les espaces sont également interdits, il est généralement conseillé de mettre un "underscore" (touche 8 de votre clavier). Exemple : var ma_Variable. Vous pouvez également lier vos mots, sans espace, en respectant la convention *Camel Case* pour plus de lisibilité.
- Le Game Maker Language est un langage qui respecte **la casse**. Cela signifie qu'il prend en compte dans l'interprétation du code les majuscules et minuscules. Ainsi, "MA_VARIABLE" et "ma_variable" seront considérées dans le programme comme deux variables différentes.



Copyright © creersonjeu.fr

## Le saviez-vous ?

Je viens de vous parler de la **convention de nommage** *Camel Case*, mais qu'est-ce que c'est au juste ?

Une convention de nommage permet aux développeurs d'adopter une écriture commune et ainsi homogénéiser leurs codes. Cela facilite la lecture ainsi que la collaboration lorsque ces derniers travaillent en équipe.

Pour respecter cette convention, il faut écrire vos ensembles de mots sans espaces. Le premier commence toujours par une minuscule, tandis que les suivants commencent par une majuscule.

Exemple :

```
1 var maVariable;
2
3
```

La convention Camel Case n'est pas obligatoire dans Game Maker.

À noter : bien que vous puissiez nommer vos variables comme vous le souhaitez, il est très important de donner des noms évocateurs à ces dernières. Cela permettra de vous retrouver dans votre code beaucoup plus facilement, et facilitera grandement la lecture de celui-ci en cas de bug ou de simple vérification !



Copyright © creersonjeu.fr

Mais revenons au sujet de notre chapitre. Concrètement, l'ajout de variables au sein de votre code se passe en deux étapes :

- La déclaration de la variable
- L'assignement d'une valeur à cette variable

Vous pouvez comme nous l'avons vu jusqu'à présent réaliser ces deux étapes en une seule fois, vous-même déclarer votre variable et lui attribuer une valeur directement.

Exemple :

```
var coucou = "Hello !";
```

Il est également possible de déclarer une variable, et de lui donner une valeur plus tard dans votre programme ! Cela peut être très utile, croyez-moi. Faites attention en revanche à bien déclarer votre variable en amont. En effet il n'est pas possible d'attribuer une valeur à une variable qui n'a pas été déclarée.

```
var i;

a = 3 ;

if (a == 3)
{
    i = 3 ;
};
```

```
a = 3 ;

if (a == 3)
{
    i = 3 ;
};

};
```

Jusqu'à présent nous attribuons à chaque fois une valeur unique et simple à notre variable. Sachez qu'il est possible d'attribuer une valeur en combinant d'autres variable ou en



Copyright © creersonjeu.fr

réalisant des opérations grâce aux opérateurs que nous avons vu auparavant. Pour cela, il suffit de **concaténer** ces dernières.

Encore un mot étrange dont la signification je vous rassure est assez simple à comprendre !

Concaténer deux variables (ou plus) signifie mettre leurs valeurs bout à bout afin de n'en former qu'une seule, grâce à l'opérateur « + » que nous avons déjà vu plus haut. On peut concaténer :

- Plusieurs nombres :

Exemple :

```
a = 3;
b = 5;

operation = a + b;

// Var operation est égal à 3 + 5, donc 8 !

```

- Plusieurs strings (chaînes de caractères) :

Exemple :

```
hello = "Bonjour";

monNom = "je m'appelle Maxime.";

presentation = hello + monNom ;

/* Cette variable sera donc égale à la chaîne de caractère
« Bonjour je m'appelle Maxime. » ! */

```

- Plusieurs types de valeurs différents.

Je vous laisse faire vos petites expériences avec la concaténation, car c'est quelque chose de très puissant !



Copyright © creersonjeu.fr

## LES SEMICOLONS

Derrière ce mot bizarre se cachent en fait … les points-virgules. Il faut savoir que dans la plupart des langages de programmation, mettre un point-virgule à la fin d'une ligne ou d'un statement est obligatoire.

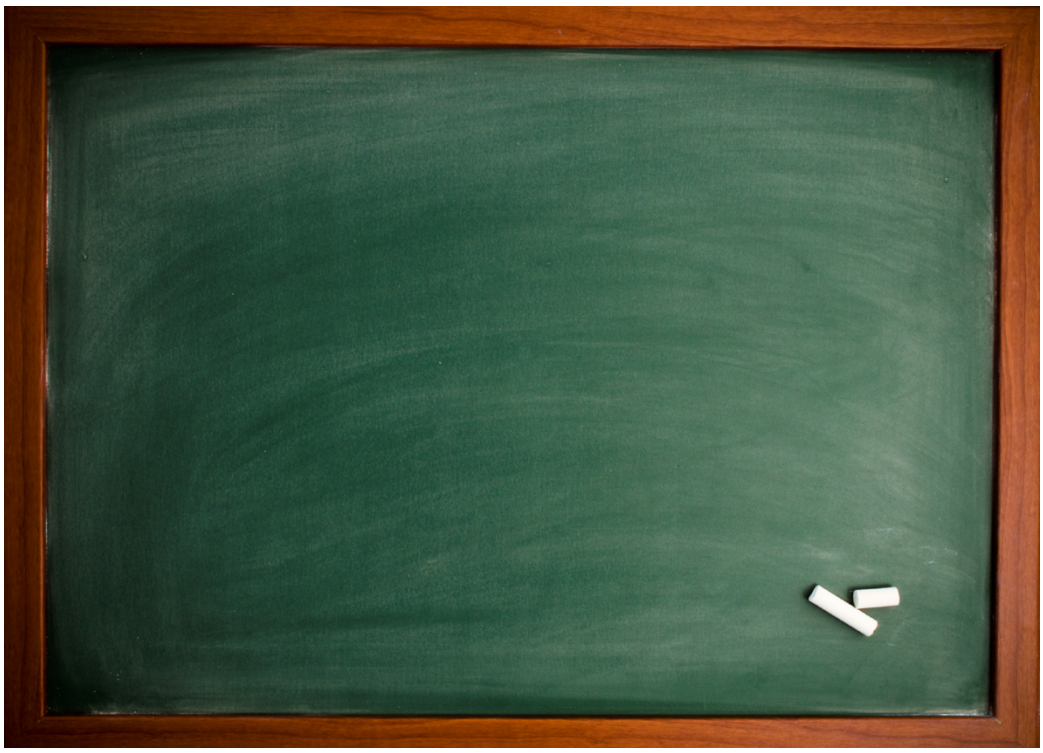
Dans Game Maker Studio, cette convention d'écriture est optionnelle. Cependant, je vous encourage vivement à la respecter. Cela vous évitera de prendre de mauvaises habitudes dans le cas où vous décideriez d'apprendre un nouveau langage par la suite.



Copyright © creersonjeu.fr

## LES ARRAYS

Les arrays qu'est-ce que c'est ? Il s'agit tout simplement … de tableaux !



*Un tableau ?*

Alors non, ce n'est pas un tableau pour écrire avec une craie ! En programmation, un array est une variable spéciale, qui permet de stocker plus d'une valeur à la fois.

Très concrètement, vous pouvez voir les tableaux comme des sortes de boîtes, qui permettent de stocker différentes valeurs de manière ordonnée.

En Game Maker Language, il existe plusieurs types d'arrays.



Copyright © creersonjeu.fr

### a. Les arrays à une dimension, ou 1 dimensional arrays

Un array à une dimension se construit de la manière suivante :

*tableau = [valeur0, valeur1, valeur2, …]*

```
1 tableau = [0, 1, 2, 3];
```

Comme vous pouvez le voir, le tableau possède un nom (ici “tableau”), suivi des valeurs qu’il contient entourées de crochets. Les valeurs contenues dans un tableau ont chacune une position numérotée. Comme le montre la capture d’écran ci-dessus, la première valeur du tableau prend **toujours** la valeur 0 !

*Un tableau peut contenir tout type de valeur. Des chaines de caractères (string), des nombres, des variables…*

À quoi vont nous servir ces valeurs de positionnement ? Tout simplement à “appeler” les valeurs contenues dans le tableau ! Pour cela il suffit simplement d’appeler dans notre code notre tableau, et mettre entre les crochets la position de la valeur désirée.

Parce qu’un exemple concret vaut toute les explications :

```
1 tableau = ["bleu", "rouge", "vert", "jaune"];
2
3 // tableau[0] sera égal à "bleu"
4
5 // tableau[3] sera égal à "jaune"
6
7 /* Rappelez-vous que la première valeur est à
8 la position 0, jaune est donc à la position 3, et non 4 ! */
9
```

*“Et si je veux créer un tableau vide ? Parce que j’ai envie de créer un tableau pour plus tard, mais que je ne sais pas encore ce que je vais mettre dedans ?”*

Et bien c’est très simple ! Il suffit d’initialiser une variable, et de lui attribuer un tableau vide, c’est à dire une paire de crochets.

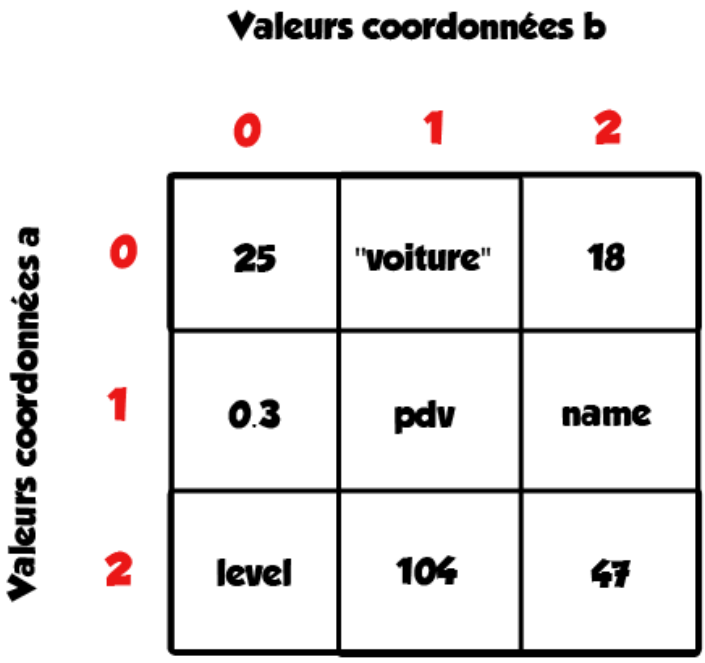
```
1 tableau = [];
```



Copyright © creersonjeu.fr

**b. Les arrays à deux dimensions, ou 2 dimensions arrays**

Un tableau à deux dimensions fonctionne plus ou moins de la même manière que ceux à une dimension que nous avons vue précédemment. Ils permettent de stocker différentes variables, à la différence que leurs coordonnées de positionnement dans le tableau ne seront plus constituées d'un seul chiffre, mais de deux. C'est encore un peu flou ? Voici un petit schéma qui devrait vous aider :




Comme vous pouvez le voir, les positions de chaque valeur contenue dans le tableau sont désormais définies par deux chiffres.

Prenons un exemple concret :

D'après le tableau ci-dessus, la valeur "voiture" est désormais disponible aux coordonnées [0, 1] !



Copyright © creersonjeu.fr

## LES COMMENTAIRES

Ecrire du code c'est bien, mais lorsque notre code devient trop long et / ou trop complexe, il faut être capable de s'y retrouver !

Si vous travaillez sur un très gros projet, il peut être utile d'ajouter entre les lignes des annotations et indications pour faciliter la lecture. De même, si d'autres personnes sont amenées à lire votre code (si vous souhaitez créer un jeu en équipe par exemple), ajoutez des commentaires pour expliquer la manière dont celui-ci est construit, cela sera très apprécié par vos collaborateurs !

Heureusement, Game Maker a prévu le coup et nous permet d'ajouter des annotations dans notre code. Celles-ci apparaîtront de couleur verte dans l'éditeur, et ne seront pas lues par le moteur pendant le jeu. Ce qui apparaît en vert ne sert qu'aux personnes qui lisent le code !

Il existe plusieurs manières d'ajouter des commentaires :

- Ajouter des commentaires courts : il suffit d'ajouter "//" en début de ligne.
  Exemple :

```
1 // Ceci est un commentaire sur une ligne
2
3
```

- Ajouter des commentaires longs, sur plusieurs lignes : dans ce cas, il faut faire débuter son commentaire par " /* " et le faire terminer par " */ ".
  Exemple :

```
1 /* Ceci est un commentaire sur
2 plusieurs lignes ! */
3
```

N'oubliez-pas, la bonne lisibilité de votre code est cruciale et peut grandement faciliter le développement de votre projet ! Prenez pour habitude de programmer en ayant toujours en tête que si quelqu'un devait un jour lire votre code, il devrait pouvoir comprendre très facilement son fonctionnement. Pour cela, ajoutez des titres aux principales parties avec " // ...", des commentaires longs avec " /* ... */ " pour expliquer le fonctionnement de passages plus complexes.



Copyright © creersonjeu.fr

# LES CONDITIONS

Une des notions les plus importantes lorsque l'on apprend à programmer !

Les conditions nous permettent de dire à Game Maker d'effectuer ou non des actions en fonction de certains paramètres. Par exemple, si mon réservoir est plein alors ma voiture avance, s'il est vide elle ne bouge pas.

Afin de pouvoir établir nos conditions, il sera nécessaire d'utiliser les opérateurs de comparaison, que nous avons vu plus haut !

Il existe plusieurs types de conditions en Game Maker Language, je vais donc vous expliquer pour chacun leur fonctionnement.

### a. Les conditions if…else

If…else signifie en français Si…sinon. Nous allons avec celle-ci, dire à Game Maker :

**Si (*notre condition*) alors {*action à réaliser*} sinon {*action à réaliser*} ;**

Comme vous pouvez le remarquer la construction se fera toujours de la même manière :

```
if (condition)
{
    action;
} else {
    action;
};
```

Prenons un exemple concret avec voiture et son réservoir. Nous allons créer une variable « reservoir » et considérer que si elle est égale à 1 le réservoir est plein, si elle est inférieure à 1 il est vide. Nous allons déterminer si la voiture peut avancer ou non en utilisant les opérateurs de comparaison que nous connaissons.



Copyright © creersonjeu.fr

```
reservoir = 1; // On part du principe que notre réservoir est plein

var niveau; // On déclare une variable niveau, qui annoncera si notre réservoir est vide, ou plein

if (reservoir == 1)
{
    niveau = "Le réservoir est plein !";
} else {
    niveau = "le réservoir est vide !";
};

```

Vous remarquerez que j'ai utilisé dans ma condition l'opérateur de comparaison « == ». Faites bien attention à ne pas le confondre avec « = » tout simple, qui sert quant à lui à attribuer une valeur à une variable !

Notez que l'on peut également se passer du « else » dans certains cas. Voyez plutôt :

```
reservoir = 1;

var niveau;

if (reservoir > 0)
{
    niveau = "Le réservoir n'est pas vide !";
};

```

Nous avons ici dit à Game Maker « Si la valeur de *reservoir* est strictement supérieure à 0, alors cela veut dire que celui-ci n'est pas vide. »

**b. Les conditions if…else if…else**

Dans l'exemple précédent nous avons établi une condition binaire. En effet nous avons dit à Game Maker « soit tu fais cette action, sinon tu fais celle là ». Mais imaginons maintenant que nous voulions introduire d'autres conditions, et donc d'autres possibilités d'actions ? C'est très simple, il suffit d'ajouter « else if » ! Cela peut se traduire en français pas « sinon si ».

Nous allons maintenant pouvoir dire à Game Maker : *« Si (ma condition) est vrai, alors tu fais cette action, sinon si (une autre condition) retourne vrai, tu fais cette action, sinon fait cette*



Copyright © creersonjeu.fr

*action. ».* Vous pouvez ajouter autant de « else if » que vous le souhaitez, les possibilités sont donc immenses !

Reprenons l'exemple du réservoir. Nous allons maintenant définir que si la variable *reservoir* est égale à 0 alors ce dernier est vide, si elle est égale à 0.5 il est à moitié plein, et si elle est égale à 1 alors cela signifie qu'il est plein.

```
reservoir = 0.5;

var niveau;

if (reservoir == 1)
{
    niveau = "Le réservoir est plein !";
} else if (reservoir == 0.5) {
    niveau = "Le réservoir est à moitié plein !";
} else {
    niveau = "Le réservoir est vide…";
};

```

Nous venons de voir les deux principaux types de conditions du Game Maker Language. Retenez les bien car elles vous serviront presque systématiquement dans vos programmes. Sachez également que vous pouvez imbriquer les conditions les unes dans les autres, ce qui multiplie une nouvelle fois les possibilités d'action pour votre programme.

Par exemple :

```
if (ma conditon)
{
    if (ma condition dans ma condition)
    {
        Mon action;
    };
};
```

**c. Les booléens dans les conditions**

Les booléens, nous l'avons vu précédemment, est un type de variable particulière qui ne peut prendre que deux valeurs distinctes : vrai, ou faux.



Copyright © creersonjeu.fr

Ce qu'il faut comprendre lorsque l'on fait appel à des conditions, c'est que l'utilisation des booléens est implicitement évoquée. En effet, lorsque l'on écrit par exemple :

```
a = 3 ;

if (a == 3)
{
    Mon action
} ;
```

Dans cet exemple, « Mon action » ne sera réalisée que si la variable « a » est bien égale à 3. Jusque-là on est d'accord ?

Cela signifie que si ma condition *a == 3* est VRAIE (vous le voyez le booléen ?) alors « mon action » sera réalisée. Si en revanche la condition est FAUSSE (par exemple *if (a == 4)*), alors « mon action » sera ignorée.

*« Mais si je veux que mon action soit réalisée si ma condition est fausse ? Comment est-ce que je fais ? »*

Ce qui est bien avec le Game Maker Language c'est que tout est prévu !

Dans Game Maker, il faut savoir qu'une action sera réalisée uniquement si la condition en amont retourne « vrai ». Il est cependant possible de contourner cette règle. Pour pouvoir réaliser une action si une condition est fausse, il va falloir dire à Game Maker :

« Si ma condition est FAUSSE (si elle est bien fausse, c'est donc VRAI), alors réalise mon action ».

Pour pouvoir faire cela, nous allons utiliser **l'opérateur de négation** « ! » que je vous ai présenté tout à l'heure. Il représente le « not » en anglais, et permet d'inverser la valeur d'un booléen. Il se place juste avant la parenthèse de notre condition.

Une fois comprise, son utilisation est très simple. Voyez plutôt :



Copyright © creersonjeu.fr

```
a = 3 ;

if !(a == 4)
{
    Mon action
} ;
```

Dans cet exemple, le point d'exclamation « ! » permet de dire à Game Maker « Si a n'est PAS EGAL à 4, alors réalise mon action. »

C'est aussi simple que ça !

Maintenant que vous avez compris comment les booléens influencent nos conditions, je vais pouvoir vous expliquer comment en combiner plusieurs.

**d. Combiner des conditions**

Vous vous souvenez des opérateurs de comparaison « && » et « || » ?

C'est grâce à ces derniers que nous allons pouvoir comparer plusieurs conditions. Ils permettent d'unifier plusieurs conditions en une seule. Si vous avez compris le principe de ces variables particulières tout à l'heure, vous devriez facilement intégrer ce qui va suivre.

Chacun de ces opérateurs possède une signification particulière :

- && signifie « ET ». Il retournera la valeur « VRAI » si les deux conditions retournent également « VRAI ».
  Exemple :

```
a = 3 ;
b = 4 ;

if ( a == 3 && b == 4)
{
    mon_Action ;
} ;

/* Les deux conditions retournent VRAI, la condition
globale retourne donc VRAI elle aussi et l'action est réalisée. */
```



Copyright © creersonjeu.fr

- || signifie « OU ». Il retournera la valeur « VRAI » si au moins une des conditions est « VRAI ».

  Exemple :

```
a = 3 ;
b = 4 ;

if ( a == 4 || b == 4)
{
    mon action ;
} ;

/* Une des deux actions retourne VRAI ( b == 4),
la condition globale retourne donc VRAI et l'action est réalisée */
```

  Notez que vous pouvez utiliser autant de ces opérateurs que vous voulez. N'oubliez cependant pas de regrouper vos conditions au sein de parenthèses.

  Exemple :

```
a = 3 ;
b = 4 ;

if (( a == 4 && b == 4) || ( a == 3 && b == 4))
{
    Mon action ;
} ;
```

**e. Les conditions Switch**

Ce type de conditions s'apparente très fortement au *if…else if…else* que nous avons vu précédemment.

Elles sont cependant bien plus simples et rapides à écrire ! Elles permettent de tester une variable en fonction de plusieurs valeurs.

Elles se construisent de la manière suivante :



Copyright © creersonjeu.fr

```
Switch (la variable que l'on veut tester)
{
    Case 1 une première condition : ce qui se passe si le résultat est VRAI ; break ;
    Case 2 une seconde condition : ce qui se passe si le résultat est VRAI ; break ;
    Case X une autre condition : ce qui se passe si le résultat est VRAI ; break ;
    Default: l'action qui est exécutée si la variable ne correspond à aucune des conditions ci-dessus ;
} ;
```

Comme vous pouvez le voir, le fonctionnement est assez simple. On écrit après le switch et entre parenthèses la variable que l'on souhaite tester. Dans les accolades on écrit sur plusieurs lignes l'ensemble des valeurs pour lesquelles on souhaite qu'une action soit réalisée (il peut y en avoir autant que l'on veut). Chaque ligne se construit de la même manière. La valeur à tester, deux points puis l'action qui doit être effectuée, un point-virgule et enfin le mot clé break ; qui permet de dire à Game Maker « Ne passe pas à la ligne suivante ».

Enfin, il est nécessaire d'ajouter une dernière ligne avec le mot clé « default ». Cela nous permet d'ajouter une action à effectuer par défaut, dans le cas ou notre variable ne correspondrait à aucune des valeurs précédemment citées.

Prenons un exemple plus concret, en reprenant notre voiture et son réservoir d'essence :

```
1  reservoir  = 1;
2  var indicateur;
3
4  switch (reservoir)
5  {
6      case (reservoir == 1) : indicateur = "Le réservoir est plein"; break ;
7      case (reservoir == 0.5) : indicateur = "Le réservoir est à moitié plein"; break ;
8      case (reservoir == 0) : indicateur = "Le réservoir est vide"; break ;
9      default : indicateur = "Tu es sur que tu as une voiture ?" ;
10 }
```



Copyright © creersonjeu.fr

## LES BOUCLES

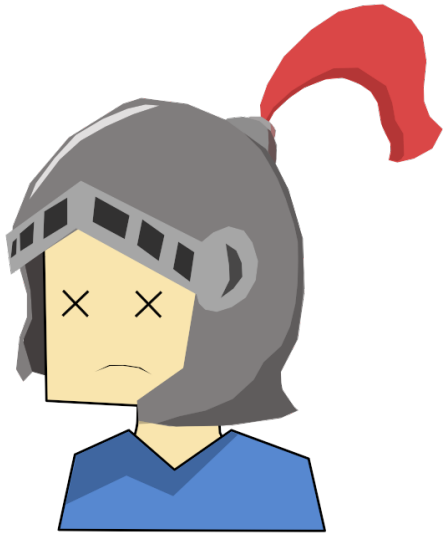
Un concept indispensable en programmation !

Vous serez probablement amenés dans votre code à vouloir réaliser plusieurs fois la même action. Face à cela, vous avez deux options : écrire autant de fois que nécessaire ladite action (cela peut être long), ou utiliser des boucles !

Le concept de boucle a été inventé pour soulager le programmeur fainéant (les programmeurs sont TOUJOURS fainéants) dans l'écriture de son code.

Prenons un exemple concret afin que compreniez bien cette notion en contexte.

Imaginons un jeu de rôle (RPG). Notre héros, un chevalier, vient d'être assailli par une horde de loups garous d'un niveau bien supérieur au sien. Après avoir utilisé tous ses sorts, toutes ses potions, celui-ci périt et est téléporté à son dernier point de sauvegarde, sa barre de vie à 0.



Nous voulons que ses points de vie se régénèrent progressivement, 1 par 1.

Comment pourrions-nous faire ? Comme je vous l'expliquais tout à l'heure nous avons deux options : l'option bête, lente et méchante ou l'option maline et rapide !



Copyright © creersonjeu.fr

Je dis toujours que l'on apprend dans la souffrance, alors on va commencer par la première option ;)

Nous pourrions écrire quelque chose comme :

```
var pointsDeVie = 0 ;

pointsDeVie += 1 ;
pointsDeVie += 1 ;
pointsDeVie += 1 ;
pointsDeVie += 1 ;
pointsDeVie += 1 ;
```

*« Euuuuhh attendez, et si jamais le héros a 100 points de vie ? Ça ne risque pas d'être un peu long ? »*

Et si ! C'est pour cela qu'il faut dans ces cas là utiliser les fameuses boucles !

Il existe plusieurs types de boucles en Game Maker Language, je vais vous montrer ici les principales :

**a. La boucle « for »**

Elle se construit de la manière suivante :

```
{
    for (<statement1> ; <expression> ; <statement2>)
        {
            <statement3>
        } ;

} ;
```

Son fonctionnement est très simple. On initialise la boucle avec le *statement 1*. L'expression qui vient ensuite est ce qui permet de faire fonctionner la boucle. Game Maker va tester si celle-ci renvoi VRAI ou FAUX (rappelez-vous, les booléens !). Si c'est FAUX, la boucle s'arrête. En revanche, si elle renvoi VRAI, alors le *statement 3* est exécuté, puis le *statement 2*, et on repart dans une nouvelle boucle.



Copyright © creersonjeu.fr

Dans notre cas admettons que notre héros, qui vient de réapparaître à l'auberge, ne possède plus qu'un point de vie. On va donc créer une variable *i*, qu'on incrémentera en fonction du nombre de point de vie du joueur. On écrira donc : **i = 1 ;**

```
1  points_de_vie = 1;
2
3  i = 1;
4
```

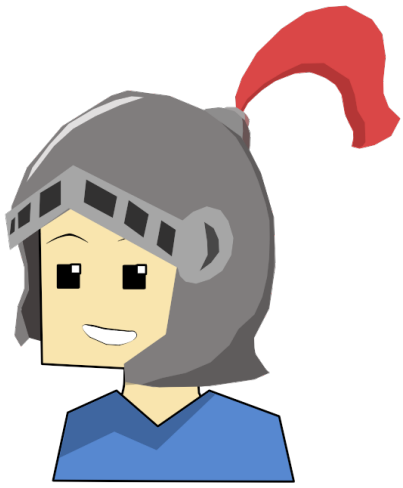
Imaginons que notre héros possède à son maximum 100 points de vie. Puisque la variable *i* possède la même valeur que la variable *points_de_vie* (vous comprendrez pourquoi par la suite), nous allons écrire dans l'expression la condition : ***i < 100 ;***

Si cela renvoi VRAI, alors notre *statement 3* sera exécuté, c'est à dire : ***points_de_vie += 1 ;***

L'objectif est de faire durer la boucle tant que *i* (et donc les points de vie) ne sont pas égaux à 100. Lorsque *i* sera égal à 100, l'expression ***i < 100 ;*** renverra FAUX, et la boucle s'arrêtera !

Avant de repartir sur une autre boucle, Game Maker va exécuter le *statement 2*. Puisque la variable *i* nous sert de repère quant à la quantité de points de vie du héros, on écrira : ***i += 1 ;***

Et voilà ! Notre boucle est en place, et notre héros va retrouver petit à petit ses points de vie, sans dépasser les 100 ! Il va pouvoir prochainement repartir au combat.





Copyright © creersonjeu.fr

Si l'on résume, notre boucle ressemblera à ça dans l'éditeur de code :

```
points_de_vie = 1 ;

for (i = 1 ; i < 100 ; i += 1)
{
    points_de_vie += 1 ;
} ;
```

**b. La boucle « while »**

Cette boucle se construit de la manière suivante :

```
while (<expression>)
{
    <statement>
} ;
```

Maintenant que vous avez compris le principe des boucles avec la boucle for, cela devrait aller bien plus vite. Reprenons l'exemple de notre héros ayant besoin de repos :

```
points_de_vie = 1 ;

while ( points_de_vie < 100)
{
    pointsDeVie += 1
} ;
```

Plutôt facile non ? Tant que l'*expression* renvoi VRAI, le *statement* est exécuté, et la boucle continue ! Inutile ici d'utiliser une variable telle que i.

Attention : si l'on ne fait pas attention, il est assez facile de créer des boucles infinies, ce qui fera planter votre jeu ! Vérifiez bien que votre boucle possède une condition qui lui permettra de s'arrêter à un moment donné.



Copyright © creersonjeu.fr

### c. La boucle do…until

Elle se construit de cette manière :

```
{
    do
    {
        <statement>
    } ;
    until (<expression>)
} ;
```

Do est le verbe « faire » en anglais. Until peut se traduire par « jusqu'à ». Le *statement* sera donc exécuté jusqu'à ce que l'expression retourne FAUX. (On peut également dire que la boucle se réalisera tant que la condition retourne VRAI).

Si l'on prend l'exemple de notre héros et de ses points de vie cela donnera :

```
1  points_de_vie = 1 ;
2
3  {
4      do
5          {
6              pointsDeVie += 1 ;
7          };
8      until (pointsDeVie < 100)
9  };
```

### d. La boucle repeat…

Cette boucle est selon moi la plus basique, elle prend la forme :

```
{
    repeat (<expression>) <statement>
} ;
```

La valeur qui suit *repeat* est le nombre de fois que le *statement* sera executé.

Dans notre exemple :

```
1  points_de_vie = 1 ;
2  repeat (99)
3      {
4          pointsDeVie += 1 ;
5      } ;
```



Copyright © creersonjeu.fr

## LES FONCTIONS

Les fonctions ! Notions incontournables quel que soit le langage de programmation, sont également présentes en GML.

Mais qu'est-ce qu'une fonction au juste ?

Une fonction est un sous-programme, qui permet d'effectuer différentes instructions simplement par son appel. Très pratique pour ne pas avoir à réécrire plusieurs fois le même code ! (Vous vous rappelez des boucles ? N'oubliez pas qu'un bon programmeur est un programmeur fainéant ;)

Une fois déclarée (il faut en revanche écrire au moins une fois cette fameuse série d'instructions) elle peut être appelée autant de fois qu'on le souhaite dans notre code.

Comment se construit une fonction ?

Une fonction possède toujours **un nom** (sensible à la casse, comme pour les variables), suivi d'un ou plusieurs **arguments** entre parenthèses. Notez que ces derniers ne sont pas obligatoires, auquel cas on écrira simplement les parenthèses vides. Il peut y avoir au maximum 16 arguments dans votre fonction.

```
// Construction d'une fonction
nom_de_ma_fonction(argument1, argument2, …, argumentX) ;
```

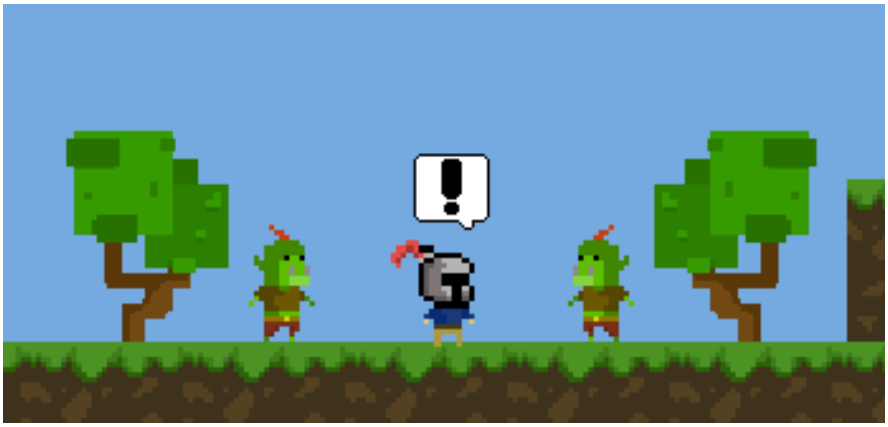
Voici un exemple concret de l'utilisation d'une fonction dans Game Maker.

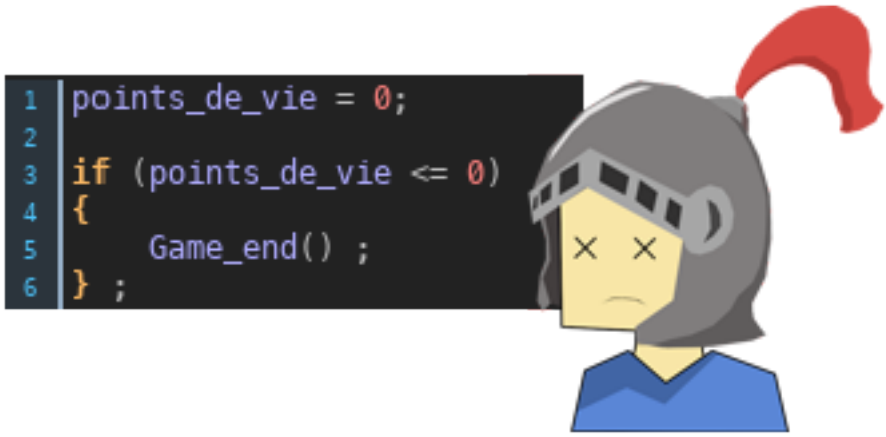
Prenons la fonction *game_end()* qui permet de stopper la partie en cours.

Mais qui voilà ? C'est notre chevalier préféré, qui vient de se faire rouster par une horde d'orcs (décidément) ! Ses points de vie sont tombés à 0, mais pourtant la partie ne s'arrête pas… Pour le laisser se reposer, nous allons donc remédier à cela.



Copyright © creersonjeu.fr






Vous pouvez le voir, nous avons dit à Game Maker « Si les points de vie du chevalier sont égaux ou inférieurs à 0 (RIP), alors le jeu s'arrête ». La fonction *game_end()* permet simplement par son appel d'exécuter une série d'instructions au préalablement rédigées. C'est quand même plus rapide et plus simple non ?

Et les arguments dans tout ça ?

Je vous ai parlé tout à l'heure d'**arguments** (paramètres en français). Pour bien comprendre, les arguments sont des variables intégrées à votre fonction, dont vous pourrez faire varier la valeur au moment de l'appel de celle-ci.

Pour vous montrer à quoi peuvent servir les arguments, reprenons notre chevalier. Celui-ci vient de trépasser et nous aimerions afficher à l'écran un game over précisant :« *nom du chevalier* est mort. »

Dans cet exemple, le *nom du chevalier* va précisément être un argument (de même que les variables *x* et *y* qui vont définir la position du texte) ! Vous pourrez le modifier à chaque fois que vous ferez appel à cette fonction. Pour cela nous allons utiliser la fonction *draw_text()*.



Copyright © creersonjeu.fr

```
nom_chevalier = "creersonjeu" ;

if (points_de_vie <= 0)
{
    draw_text(x, y, nom_chevalier + "est mort… Game Over !") ;
} ;
```

Il existe dans Game Maker des fonctions préinstallées dans le logiciel (built-in functions, comme *game_end()* ou *draw_text()* par exemple) que vous pouvez utiliser à votre guise. Ces dernières sont très utiles, nous en avons d'ailleurs utilisées plusieurs lors de la réalisation des tutoriaux sur la **chaîne YouTube Creersonjeu.fr** !

Il est également possible de créer vos propres fonctions, en créant des scripts.

**Les scripts**

Comme je viens de vous l'expliquer, les scripts sont dans Game Maker des fonctions que vous pouvez rédiger vous-même. Ils sont très utiles pour ne pas avoir à réécrire plusieurs fois les mêmes lignes d'instructions.

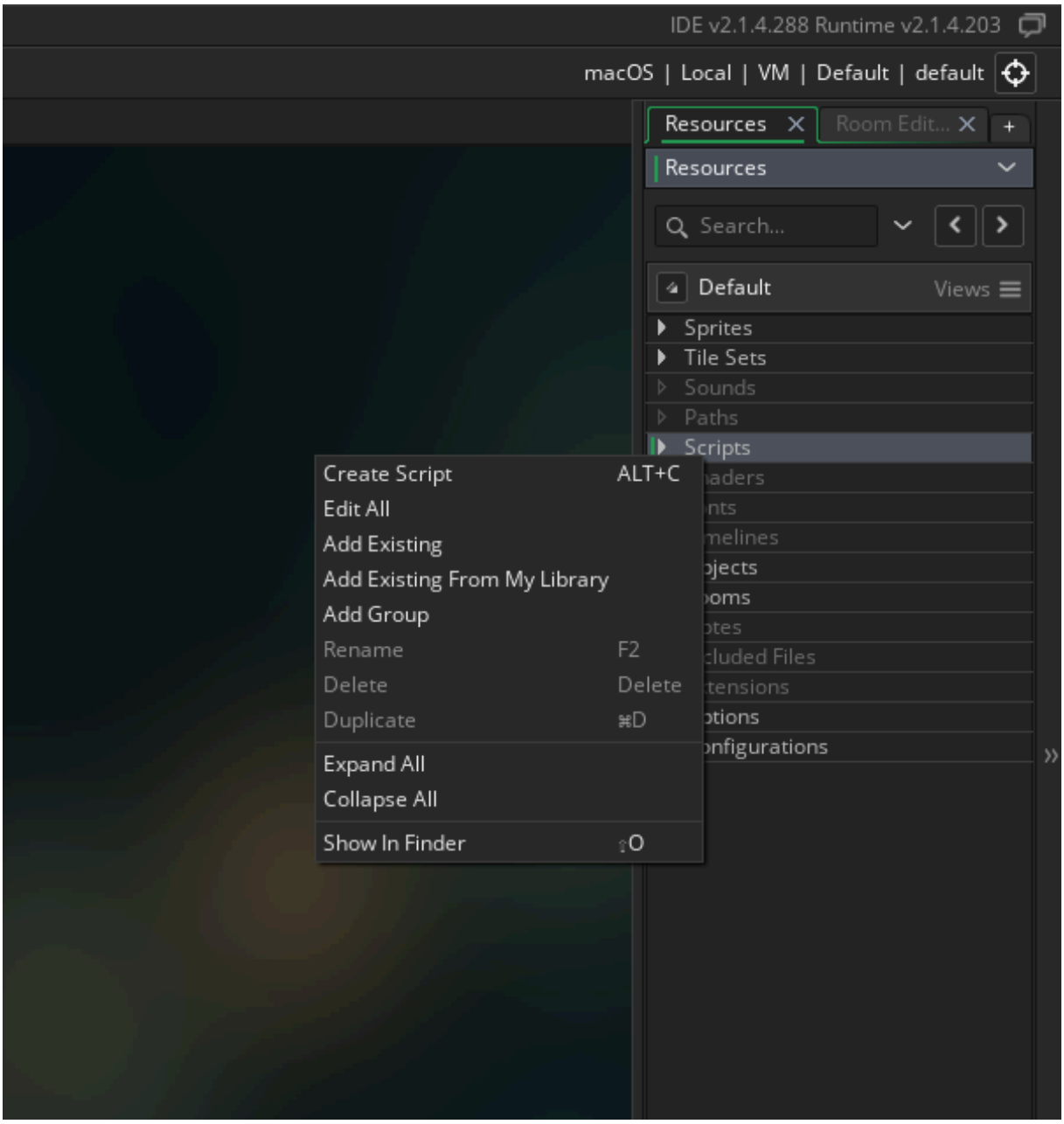
Tout comme les fonctions, ils peuvent posséder au maximum 16 arguments.

On les appelle de la même manière que les fonctions (le nom suivi d'entre parenthèses les arguments s'il y en a) et sont accessibles autant de fois que vous le souhaitez, à partir de n'importe quel objet de votre programme.

Pour créer vos propres scripts, il vous suffit de vous rendre dans l'onglet dédié sur le tableau de bord Game Maker, et sélectionner *Create Script.*



Copyright © creersonjeu.fr




Nous allons maintenant créer ensemble un petit script. Suite à sa déconvenue avec les orcs, notre chevalier a pu sortir du jeu et réapparaitre à l'auberge grâce à notre intervention. Ses points de vie se régénèrent tranquillement 1 par 1 (sujet abordé au chapitre des boucles). Pour passer le temps, il décide d'aller faire un tour à la taverne histoire de se rafraîchir avec une bonne boisson. Malheureusement pour lui, une bagarre éclate pendant qu'il se désaltère, et ses points de vie ne sont pas complètement restaurés !

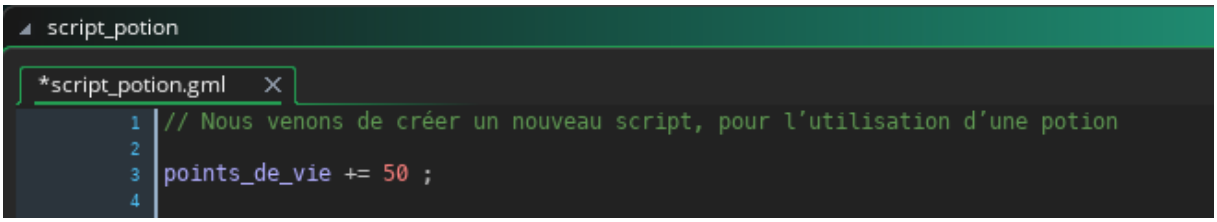
Pour l'aider, nous allons rédiger un petit script qui décrira l'effet de l'utilisation d'une potion.



Copyright © creersonjeu.fr

Remarque : Pour notre exemple, nous allons considérer que si la variable *utilisation_potion* est égale à 1, alors c'est que le chevalier est en train d'en utiliser une.

Création d'un nouveau script que l'on nomme « script_potion »…




… auquel nous allons faire appel dans notre objet chevalier, dans son event step !

```
var utilisation_potion;

if (utilisation_potion == 1)
{
    script_potion() ;
} ;
```

Comme vous pouvez le voir, nous avons créé un script potion qui, lorsqu'il est appelé, incrémente de 50 la variable *points_de_vie*. Nous avons donc indiqué dans notre objet chevalier que lorsqu'une potion est utilisée, nous faisons appel au script *script_potion().*

C'était ici un exemple très simple, mais vous pouvez faire des choses bien plus complexes avec ce système !


Copyright © creersonjeu.fr

# LES FONCTIONS ET VARIABLES PRÉ-INTÉGRÉES DANS GAME MAKER STUDIO 2

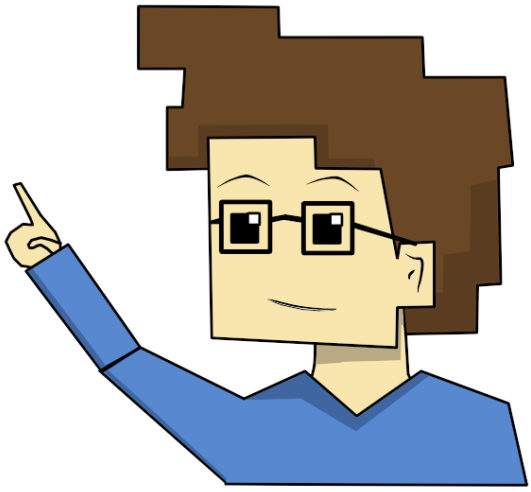


Copyright © creersonjeu.fr

Je vous en ai déjà parlé précédemment, il existe dans Game Maker un nombre très important de fonctions pré-intégrées. Ces dernières sont extrêmement utiles pour la création de vos jeux, l'avantage étant que vous n'avez plus qu'à faire appel à elles puisque celles-ci ont été préprogrammées !

Il existe également certaines variables prédéfinies qui s'avèreront extrêmement utiles et vous permettront de réaliser de nombreuses choses pour votre jeu.

Je vous préviens tout de suite il en existe une multitude et il serait beaucoup trop long de toutes les lister et de vous expliquer leur fonctionnement dans ce livre. C'est pour cela que je vous ai fait une sélection (non exhaustive) des fonctions et variables les plus utiles. Tout débutant se doit de les connaître pour démarrer la programmation de son jeu sur de bonnes bases. Si vous souhaitez aller plus loin dans la découverte des fonctions et variables pré-intégrées, je vous encourage vivement à aller les consulter sur la documentation officielle Game Maker Studio 2 à cette adresse : **Documentation Officielle.**





Copyright © creersonjeu.fr

## FONCTIONS ET VARIABLES LIÉES AUX ROOMS

Bien que cet ouvrage ne soit pas dédié à l'utilisation de Game Maker Studio, je vais quand même vous rappeler ce qu'est une room.

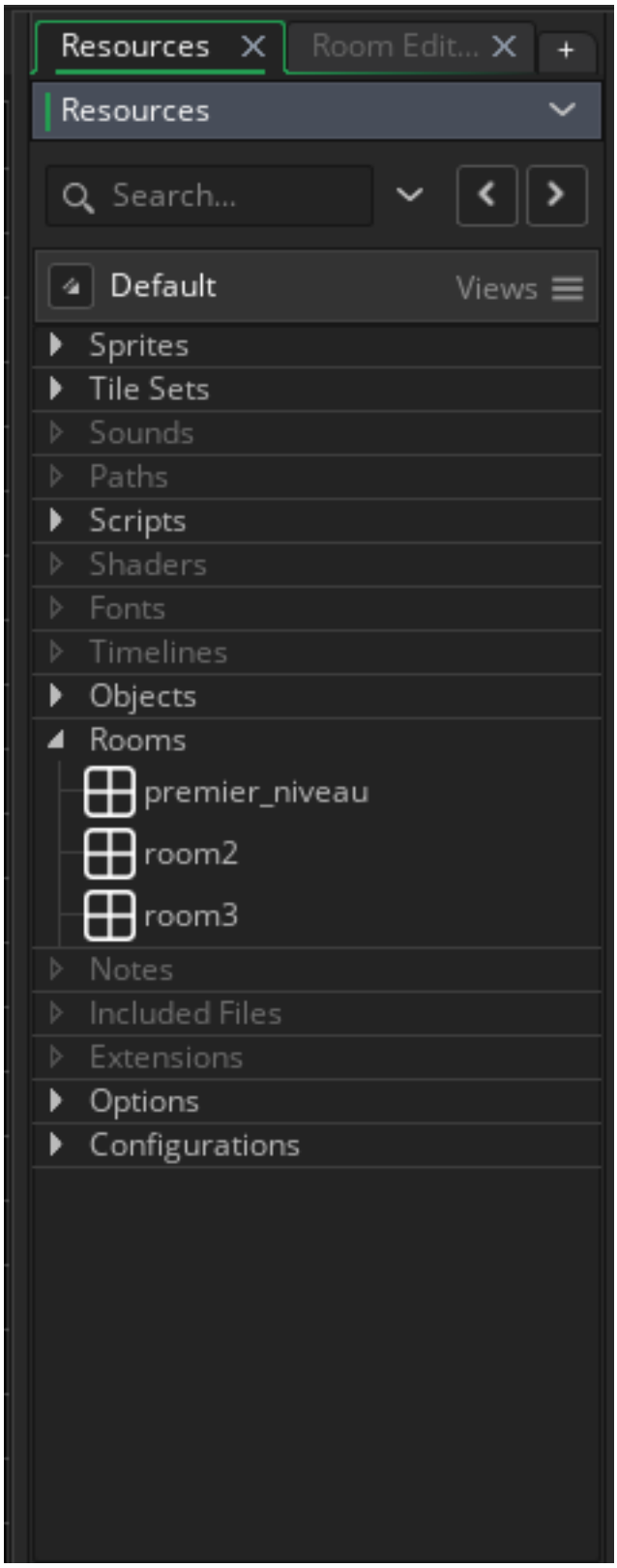
Une room est l'espace dans lequel vont évoluer les différents objets de votre jeu. Elle est divisée en différents layers, permettant de classer et de positionner sur différents plans les objets, tilesets, etc…

C'est également dans cette dernière que l'on ajoute une ou plusieurs caméras afin de pouvoir afficher à l'écran ce qu'il se passe dans votre jeu ! La vision de la caméra est représentée dans Game Maker Studio par le *viewport*.

Vos rooms sont classées dans une liste sur votre tableau de bord Game Maker. Faites bien attention à leur ordre, car il sera très important pour la suite. Lorsque vous lancer votre jeu, celui-ci s'ouvre systématiquement sur la première room de votre liste.



Copyright © creersonjeu.fr




Voici maintenant un ensemble d'actions réalisables avec les fonctions et variables dédiées aux rooms.



Copyright © creersonjeu.fr

### a. Récupérer la hauteur (height) et la largeur (width) de votre room

Ces valeurs sont accessibles via les variables **room_height** et **room_width.**

Elles vous permettent à tout moment de récupérer les dimensions d'une room, et pourquoi pas de les changer !

```
room_height;

room_width;
```

Exemple :

```
draw_text(room_width / 2, room_height  / 2, "Hello !") ;

/* Les variables nous permettent de récupérer très facilement les dimensions de la room,
et d'écrire au centre de celle-ci le texte "Hello !" */
```

Dans cet exemple, nous avons récupéré grâce aux variables *room_width* et *room_height* les dimensions de notre room, puis nous avons divisé par deux ces valeurs afin de placer le texte au centre de cette dernière.

### b. Se déplacer de room en room au cours du jeu

Les différents tableaux de votre jeu sont organisés dans Game Maker en différentes rooms. Problème : comment passer d'un tableau à l'autre pendant le jeu ? C'est en réalité très simple grâce aux fonctions :

- **room_goto(*index*) ;**

  Cette fonction permet de se déplacer dans une nouvelle room au cours du jeu. Pour cela il suffit simplement de préciser son *index* entre parenthèses. Mais qu'est-ce que l'index ? C'est tout simplement le nom de la room à laquelle vous souhaitez accéder !



Copyright © creersonjeu.fr

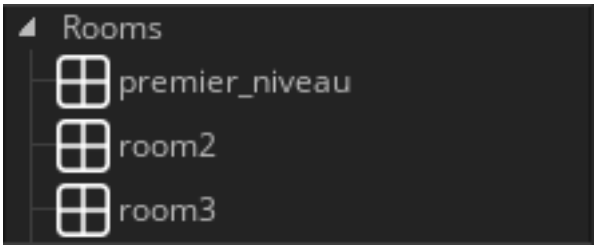
Exemple :

```
/* Si les points de vie du chevalier tombent à 0,
alors on veut qu'il se téléporte à la room « room_auberge » */

if (points_de_vie == 0)
{
    room_goto(room_auberge) ;
} ;
```

- **room_goto_next() ;**

Cette fonction permet de passer directement à la room suivante. Faites donc bien attention à ce qu'elle existe dans votre arborescence, sinon Game Maker vous retournera un message d'erreur !




Si la room en cours est « premier_niveau », alors faire appel à la fonction room_goto_next() nous amènera directement à la room « room2 » !

- **room_goto_previous() ;**

De la même manière que *room_goto_next()*, cette fonction vous permet de revenir à la room précédente de votre liste.

- **room_restart() ;**

Permet de réinitialiser la room en cours. L'ensemble des objets, scripts, et timelines présents dans cette dernière seront réinitialisés également.



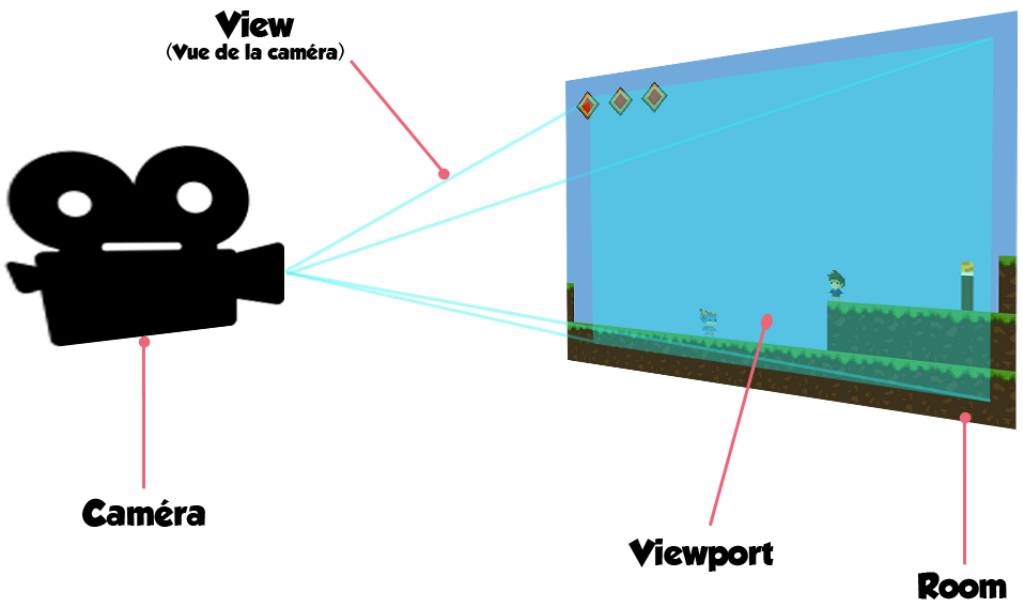
Copyright © creersonjeu.fr

## FONCTIONS ET VARIABLES D'AFFICHAGE (CAMERAS, VIEW, VIEWPORTS)

Lorsque vous créez une nouvelle room, il est possible de configurer les différentes *cameras* et *viewports* de cette dernière. Laissez-moi vous expliquer concrètement à quoi cela correspond.

Les caméras dans Game Maker permettent de filmer une partie de votre room. Lorsque regardez un film par exemple, vous ne voyez pas l'ensemble du décor n'est-ce pas ? Vous voyez seulement **une partie** de celui-ci, ce qui permet au spectateur de se concentrer sur l'action en cours. Et bien c'est exactement la même chose dans Game Maker !

Il est possible de configurer une infinité de caméras, en revanche seulement 8 peuvent être actives en même temps (vous l'imaginez la partie multi-joueurs à 8 sur le même écran ?). Elles permettent au joueur de se focaliser sur une partie de la room seulement. Ce que « filme » la caméra s'appelle la *view*. Cette *view* va ensuite être retranscrite sur le *viewport*, et c'est ce dernier qui sera affiché sur l'écran du joueur.






Copyright © creersonjeu.fr

Voici les différentes fonctions et variables permettant d'influer sur ces différents éléments :

**a. Créer et détruire une caméra**

- **camera_create() ;**

Cette fonction permet de créer une nouvelle camera, vide. C'est-à-dire que sa view ne sera pas configurée.

Exemple :

```
camera_view[1] = camera_create() ;
```

- **camera_destroy(camera_id) ;**

Permet de détruire une caméra, en précisant son ID (son numéro) obtenue sur le tableau de bord Game Maker (ou si vous en avez créé une avec la fonction *camera_create()* )

**b. Récupérer la largeur et la hauteur (de la view) de votre caméra**

- **camera_get_view_width(camera_id) ;**

Peut par exemple servir à stocker la valeur de la largeur d'une caméra dans une variable.

- **camera_get_view_height(camera_id) ;**

La même fonction, mais cette fois-ci pour récupérer la hauteur !

**c. Activer ou désactiver un viewport**

- **view_visible[*numéro du viewport*] ;**

Il vous suffit pour cela d'inscrire entre les crochets le numéro du viewport et de lui assigner la valeur *true*.

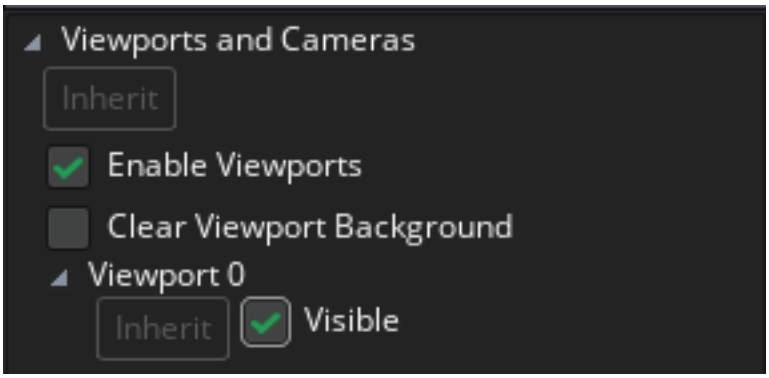


Copyright © creersonjeu.fr

Exemple :

```
1 view_visible[0] = true ;
```

Cela correspond concrètement à cocher cette case sur notre tableau de bord :




**d. Récupérer la position sur l'axe des x et des y d'un viewport**

- **view_get_xport(*numéro du viewport*) ;**
- **view_get_yport(*numéro du viewport*) ;**

**e. Récupérer la largeur (width) et la hauteur (height) d'un viewport**

- **view_get_wport(*numéro du viewport*) ;**
- **view_get_hport(*numéro du viewport*) ;**

**f. Assigner une nouvelle position sur l'axe des x et des y à un viewport**

- **view_set_xport(*numéro du viewport, nouvelle position sur l'axe des x*) ;**
- **view_set_yport(*numéro du viewport, nouvelle position sur l'axe des y*) ;**



Copyright © creersonjeu.fr

**e. Assigner une nouvelle valeur à la largeur et la hauteur d'un viewport**

- **view_set_wport(*numéro du viewport, nouvelle largeur)* ;**

- **view_set_hport(*numéro du viewport, nouvelle hauteur*) ;**

**f. Assigner une caméra à un viewport**

- **view_set_camera(*numero du viewport, ID de la camera*) ;**



Copyright © creersonjeu.fr

# FONCTIONS ET VARIABLES DE GESTION DES DÉPLACEMENTS, COLLISIONS

### a. Gestions des déplacements

Il faut savoir que dans Game Maker, le temps qui s'écoule est composé de « steps ». Les mouvements sont calculés de manière générale en *pixel par step*.

**Les variables**

- **hspeed ;**

  Permet de définir la vitesse en pixel sur l'axe horizontal (l'axe des x donc) de n'importe quel objet.

- **vspeed ;**

  Permet de définir la vitesse en pixel sur l'axe vertical (l'axe des y donc) de n'importe quel objet.

Attention : dans Game Maker, l'axe vertical est inversé. Les points positifs seront donc vers le bas !

- **direction ;**

  Permet de définir la direction du mouvement d'un objet, en degré. Selon la *documentation officielle Game Maker*, 0° oriente la direction vers la droite, 90° vers le haut, 180° vers la gauche et 270° vers le bas.

Attention : cette variable ne peut être utilisée que si l'objet possède une variable speed différente de 0 (c'est-à-dire qu'il est en mouvement).

- **speed ;**

  Permet de définir la vitesse de déplacement d'un objet. S'utilise en complément de la variable **direction**. La vitesse sera calculée en pixel par step.



Copyright © creersonjeu.fr

Si vous attribuez une valeur négative à cette variable, l'objet ira dans la direction opposée à celle définie par la variable **direction**.

- **friction ;**

Une variable très intéressante ! Elle permet d'ajouter un coefficient de friction aux déplacements d'un objet. Elle ne peut s'utiliser que si l'on a au préalable attribué une valeur différente de 0 à la variable speed de ce même objet.

Comment ça marche ?

C'est très simple. Imaginons que la variable speed de votre objet soit égale à 5 et la friction à 1. Je vous l'ai expliqué tout à l'heure, l'objet avancera de 5 pixels à chaque step. Avec la variable friction définie, on va soustraire à chaque step sa valeur (1) à la valeur speed (5). À chaque step la valeur speed va diminuer de 1 ! (5…4…3…2…1…0). L'objet va donc ralentir progressivement et s'arrêter au bout de 5 steps ! Pratique lorsque vous programmez le comportement d'une voiture par exemple.

- **gravity ;**

Encore une variable très très utile ! Elle permet d'appliquer une force sur un objet, simulant un effet de gravité. Quoi de mieux pour un platformer ? ;) La direction de la force sera à définir avec la variable **gravity_direction**. La force nouvellement appliquée va influencer la force exercée par la variable **speed**.

- **gravity_direction ;**

Permet de définir la direction de la force appliquée via la variable **gravity**.

De la même façon que la variable direction**,** 0° oriente vers la droite, 90° vers le haut, 180° vers la gauche et 270° vers le bas.

Attention : si la valeur de la variable **gravity** est égale à 0, alors cette variable ne sert à rien !



Copyright © creersonjeu.fr

### b. Gestion des collisions

Il existe plusieurs types de fonctions, permettant de vérifier différents types de collisions. Je vais vous montrer ici uniquement certaines fonctions de bases vous permettant de tester les collisions simples.

#### Les fonctions

- **place_meeting(x, y, l'objet que l'on souhaite tester pour la collision) ;**

Celle-ci doit sembler très familière à tous ceux qui me suivent sur YouTube ! En effet, nous l'avons utilisé plusieurs fois dans les tutoriaux. Elle permet de tester si un objet entre en collision avec un autre à une position donnée.

Notez que les valeurs x et y entre parenthèses sont calculées par rapport à la position de l'objet qui fait appel à la fonction.

Exemple :

```
1 if (place_meeting(3, 0, obj_enemy))
2 {
3     speed = 0 ;
4 } ;
```

Dans cet exemple, Game Maker va tester s'il y a une collision entre notre objet chevalier et un objet « obj_enemy » à la position +3 (par rapport au centre de l'objet) sur l'axe des x. Si c'est le cas, alors le chevalier s'arrête.



Copyright © creersonjeu.fr

- **position_destroy(x, y) ;**

Cette fonction permet de détruire tout objet dont le masque de collision se trouve au contact de la position donnée entre parenthèse.



Copyright © creersonjeu.fr

## FONCTIONS DE CONTRÔLE DE PARTIES

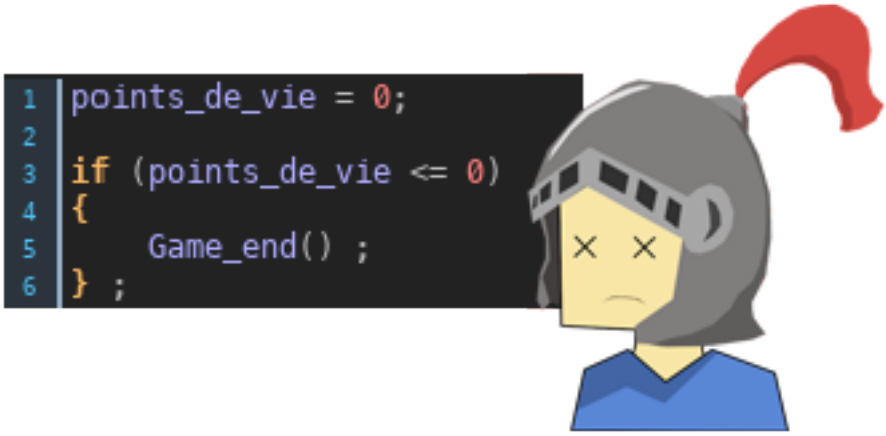
De la même manière que l'on peut influer sur le comportement des objets, sprites etc de notre jeu, certaines fonctions préinstallées permettent de contrôler le jeu en lui-même. En voici quelques-unes, qui vous permettront de maitriser certains aspects basiques de tout jeu.

**Les fonctions**

- **game_end() ;**

Cette fonction permet tout simplement d'arrêter le jeu en cours d'exécution. Rappelez-vous, nous l'avons utilisé tout à l'heure pour stopper la partie lorsque les points de vie du chevalier étaient tombés à 0.

Exemple :




- **game_restart() ;**

Cette fonction permet quant à elle de redémarrer la partie en cours, comme si nous démarrions le jeu pour la première fois.

Attention : les **variables globales** (qui s'écrivent de cette manière : *global.nom_variable*) ne seront en revanche pas réinitialisées, il faudra le faire vous-même !



Copyright © creersonjeu.fr

Beaucoup d'entre vous me l'ont demandé en commentaire sur YouTube : comment faire un système de sauvegarde ? Bien qu'un système complexe (et complet) soit très long à expliquer, voici déjà deux fonctions préintégrées basiques, qui vous permettront de faire très rapidement un système de save/load pour votre jeu.

- **game_save (« *nom de la sauvegarde* ») ;**

Cette fonction vous permet comme son nom l'indique de sauvegarder sous un fichier **.dat** votre jeu. La sauvegarde ne sera pas très poussée et ira se placer dans le dossierdu jeu. Elle gardera simplement en mémoire la room en cours ainsi que la valeur des variables, mais c'est déjà un bon début.

Attention : veillez à bien écrire l'extension .dat, ainsi que le nom de la sauvegarde entre guillemets.

Exemple :

```
1 if (keyboard_check_pressed(ord("P")))
2 {
3     game_save("chevaliersave1.dat") ;
4 } ;
```

Si la touche P du clavier a été appuyée, alors on crée une sauvegarde ayant pour nom « chevaliersave1.dat ».

- **game_load (« *nom de la sauvegarde à charger* ») ;**

  Vous vous en doutez, cette fonction vous permettra de charger le fichier de sauvegarde créé avec **game_save() ;**



Copyright © creersonjeu.fr

# FONCTIONS AUDIO

Parce qu'un jeu avec un très bon gameplay, de beaux graphismes, une histoire incroyable c'est bien, mais c'est quand même mieux avec des bruitages et de la musique ! Voici quelques fonctions qui vous permettront d'ajouter vos premiers effets sonores à votre jeu.

Notez que pour ajouter une ressource de type *sound* à votre jeu, il faut procéder comme pour les sprites. Allez dans le menu de droit, clic droit sur l'onglet « sound » et ajoutez un nouveau son !

**Les fonctions**

- **audio_play_sound(*nom du fichier audio, ordre de priorité, jouer en boucle ou non*) ;**

  La fonction audio par excellence ! C'est grâce à elle que vous pourrez ajouter une musique qui fait peur dans le manoir du dernier niveau, ou une musique entraînante pour le premier !

  Son utilisation est assez simple. Trois valeurs sont à renseigner entre parenthèses.

  La première est le nom du fichier son que vous avez ajouté à votre projet. La seconde est l'ordre de priorité du son à jouer. Je m'explique. Sur l'ensemble des sons qui vont être joués au cours de votre partie, Game Maker ne va pas pouvoir tous jouer en même temps. C'est pourquoi il faut leur donner à chacun un numéro, afin d'indiquer au programme quel son est à jouer en priorité. Lorsqu'un nouveau son est activé, tous les sons avec une priorité inférieure s'arrêteront.

  Enfin, la dernière valeur est la valeur loop. Deux choix s'offre à vous : *true* ou *false*. Si vous optez pour la valeur *true*, alors le son sera joué en boucle. Si c'est false, il ne sera joué qu'une seule fois et s'arrêtera ensuite.



Copyright © creersonjeu.fr

Exemple :

```
if (mon_score == 1)

{
    audio_play_sound(sound_win, 3, false) ;
} ;
```

Si le joueur a un score égal à 1, alors Game Maker va jouer un son de victoire, avec une priorité de 3 (tous les sons avec une priorité inférieure vont s'arrêter). Le bruitage ne sera en revanche joué qu'une fois, puisque la loop est false.

- **audio_pause_sound(*nom du fichier audio*) ;**

Très pratique lorsque vous réalisez un menu pause, cette fonction permet de stopper une musique ou un son en train d'être joué. Indiquez simplement entre parenthèses le nom du fichier son à mettre en pause. Si vous décidez de remettre sur play ce même son plus tard, il reprendra là où il s'était arrêté.

- **audio_stop_sound(*nom du fichier audio*) ;**

Fonctionne de la même manière que la fonction **audio_pause_sound() ;** à la différence que cette fois-ci le son ne sera pas mis en pause mais stoppé. Si vous décidez de le rejouer plus tard, il reprendra du début.



Copyright © creersonjeu.fr

# FONCTIONS ET VARIABLES LIÉES AUX INSTANCES

Je vais maintenant vous présenter un ensemble de variables et fonctions vous permettant de contrôler de manière plus efficace les instances de votre jeu.

Pour rappel, les instances sont les différents objets ayant des interactions dans votre jeu (les objets présents dans votre room notamment).

## Les fonctions

### a. Créer une nouvelle instance, sur un layer donné

- **instance_create_layer(*x, y, nom du layer, nom de l'objet*) ;**

Cette fonction permet de créer sur un layer une nouvelle instance à partir d'un objet. Elle fonctionne très simplement : il suffit de donner une position x et y pour définir les coordonnées auxquelles celle-ci sera créée, donner le nom du layer de destination, et écrire le nom de l'objet à faire apparaître. Elle peut être très utile lorsque l'on veut faire apparaître des balles de pistolets, des flèches par exemple.

Notez que laisser x et y sans valeur fera apparaître la nouvelle instance à l'origine de l'objet qui a fait appel à elle.

Exemple :

```
1 if (keyboard_check_pressed(ord("P ")))
2 {
3     instance_create_layer(x, y, layer_player, obj_bullet) ;
4 } ;
```

*Lorsque le joueur appuie sur la touche « P », son pistolet tire des balles, représentées par l'objet « obj_bullet ».*



Copyright © creersonjeu.fr

### b. Détruire une instance

- **Instance_destroy(*nom de l'instance à détruire)* ;**

On l'a vu précédemment, créer des instances peut être très pratique si l'on veut par exemple générer des projectiles dans notre jeu. En revanche, il faut pouvoir détruire ces mêmes projectiles. Si par exemple votre balle de pistolet touche un mur ou un ennemi, on veut qu'elle disparaisse. C'est avec cette fonction que vous pourrez le faire.

Il existe trois manières d'utiliser cette fonction.

**Manière numéro 1 :** Laisser les parenthèses vides. Cela aura pour effet de détruire l'instance qui a fait appel à cette fonction.

Exemple :

```
1 if ( points_de_vie == 0)
2 {
3     instance_destroy() ;
4 } ;
```

*Nous sommes dans notre objet Enemy. Si ses points de vies tombent à 0, alors l'instance est détruite et l'objet disparaît de la carte.*

**Manière numéro 2 :** Ecrire le nom d'un objet entre parenthèse. Cela aura pour effet de détruire sur la map l'ensemble des objets portant ce nom.

Exemple :

```
1 if (superpouvoir == 1)
2 {
3     instance_destroy(obj_enemy) ;
4 } ;
```

Si le superpouvoir est activé, tous les ennemis représentés par l'objet « obj_enemy » seront détruits en même temps !



Copyright © creersonjeu.fr

**Manière numéro 3 :** Lorsque vous testez si une collision a lieu, mettre « other » entre les parenthèses permet de détruire l'autre instance que celle qui fait appel à la fonction.

Exemple :

```
1 if (place_meeting(x, y, obj_orc))
2 {
3     instance_destroy(other) ;
4 } ;
```

*Nous testons si une collision à lieu entre le chevalier et un orc. Si les deux se touchent, alors l'orc sera détruit. On teste cette collision dans l'event step de notre objet chevalier.*



Copyright © creersonjeu.fr

# FONCTIONS DE DESSIN

Il est possible dans Game Maker de dessiner par-dessus votre scène grâce à diverses fonctions : des formes, lumières, du texte… Ce que vous dessinerez n'aura pas d'interactions avec les instances de votre jeu, en revanche cela vous servira pour améliorer l'immersion (via des effets spéciaux par exemple) ainsi qu'ajouter des aides pour le joueur (en dessinant un GUI comme une barre de vie).

Vous souhaitez ajouter un cycle jour/nuit à votre jeu de survie ? Ajouter une bulle de texte pour faire parler les PNJ de votre RPG ? Voici quelques fonctions qui devraient vous être fortement utiles ;)

**Les fonctions**

Il faut savoir que l'ensemble des fonctions permettant de dessiner commencent toujours par « draw ».

Attention ! Veillez à toujours écrire ces dernières dans l'event « draw » de vos objets, sinon cela ne fonctionnera pas !

**a. Dessiner du texte**

- **draw_text(*x, y, « texte »*) ;**

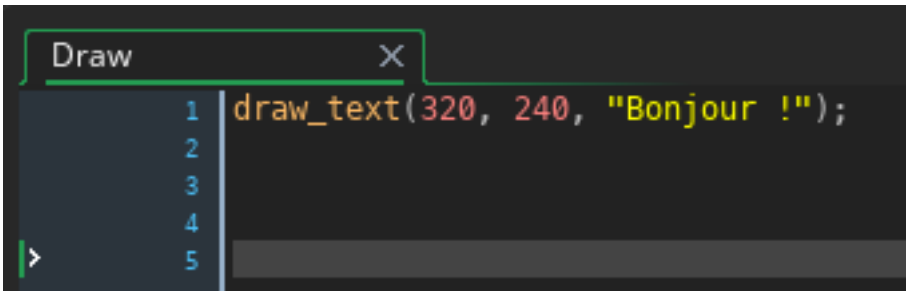
  La plus basique des fonctions permettant d'écrire du texte. Il vous suffit d'écrire les coordonnées d'affichage en x et y, puis entre guillemets le texte à écrire.

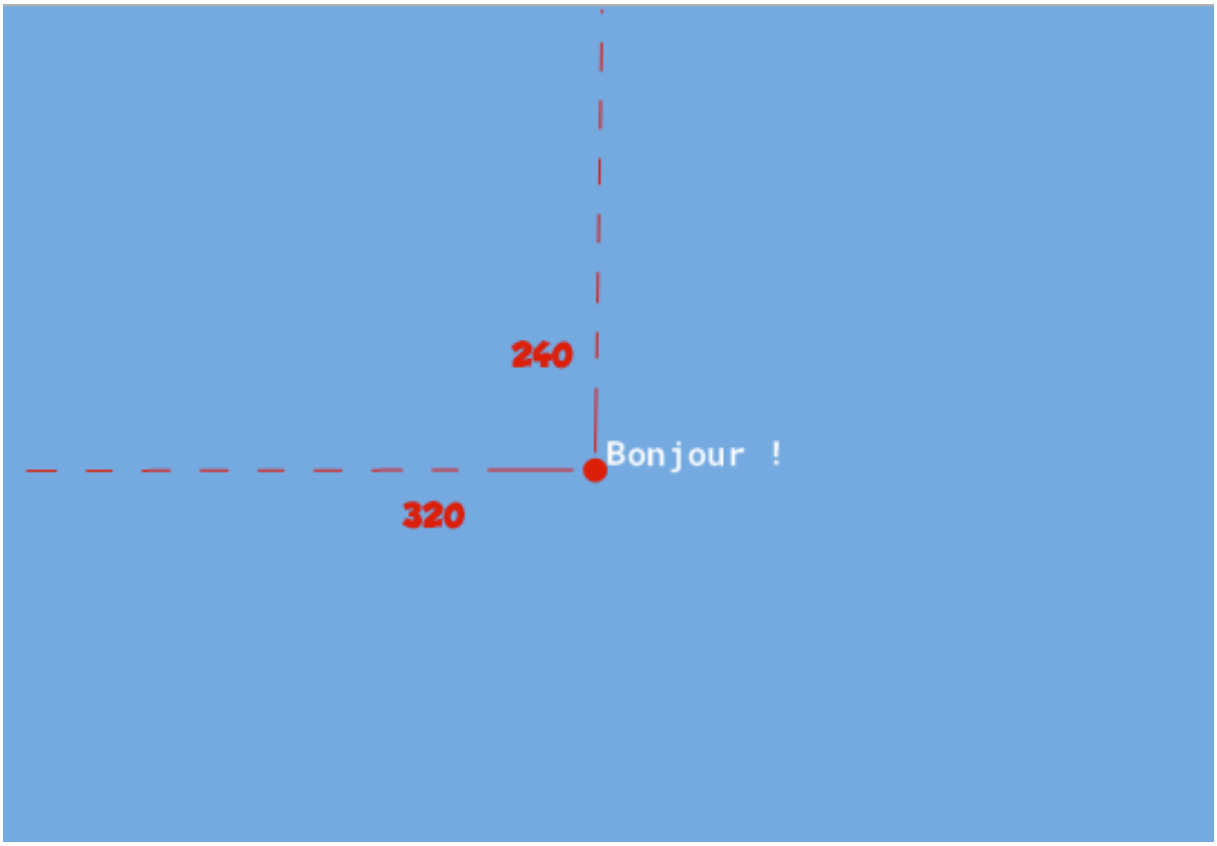
  Attention : cette fonction doit être appelée dans l'event « draw » de vos objets !

  Exemple :



Copyright © creersonjeu.fr




Ecrire du texte c'est très bien, mais si l'on souhaite changer la couleur, la police d'écriture utilisée, l'alignement… ? Game Maker à heureusement pensé à tout, et il est possible d'utiliser plusieurs fonctions spécifiques afin de personnaliser tout ça.

- **draw_set_font(*police d'écriture à utiliser*) ;**

  Il faut au préalable avoir ajouté un fichier font dans le dossier spécifique de votre tableau de bord. Il vous suffit ensuite d'écrire cette fonction avant votre fonction **draw_text()**. Tout le texte qui suivra utilisera ainsi cette font.

- **draw_text_colour(*x, y, « texte », c1, c2, c3, c4, alpha*) ;**



Copyright © creersonjeu.fr

Woa, il y a beaucoup de paramètres dans cette fonction non ? Ne vous inquiétez pas, ce n'est pas aussi compliqué que ça en a l'air. Détaillons tout ça ensemble.

Cette fonction s'utilise de la même manière que **draw_text()** à la différence que l'on peut avec celle-ci déterminer la couleur et la transparence du texte à dessiner.

Imaginez que votre texte est contenu par un rectangle invisible, et que l'on puisse attribuer à chacun de ses côtés une couleur. *C1* correspondra au coin supérieur gauche, *c2* au coin supérieur droit, c3 au coin inférieur droit et *c4* au coin inférieur gauche.

Il ne vous reste plus qu'à remplacer chacune de ces variables par une couleur afin de colorer votre texte. Vous pouvez bien entendu mettre la même couleur afin que le texte soit uni ou mettre des couleurs différentes afin de créer un dégradé. Voyez plutôt :

```
1 draw_text_colour(320, 240, "Bonjour", c_blue, c_orange, c_blue, c_orange, 1);
2
```

- **draw_set_halign(*alignement à préciser*) ;**

Voici une fonction bien pratique lorsque l'on dessine du texte sur Game Maker. Elle va de pair avec **draw_set_valign()** que nous verrons juste après. Celle-ci vous permet d'aligner votre texte sur l'axe horizontal. Concrètement, cela va influer sur le sens d'écriture du texte (toujours en fonction des coordonnées que vous avez renseigné en x et y).



Copyright © creersonjeu.fr

Pour ceux qui utilisent Word, cela correspond à ces boutons :

Il vous suffira de préciser l'une de ces 3 constantes dans les parenthèses :

- *fa_left* : permet d'aligner le texte à gauche des coordonnées d'écriture
- *fa_center* : permet de centrer le texte aux coordonnées d'écriture
- *fa_right* : permet d'aligner le texte à droite des coordonnées d'écriture

Par défaut lorsque vous écrivez du texte, *fa_left* est appliqué.

- **draw_set_valign(*alignement à préciser*) ;**

Comme nous l'avons vu pour **draw_set_halign()**, cette fonction permet de déterminer l'alignement vertical du texte, par rapport à ses coordonnées d'écriture renseignées en x et y.

Voici les constantes à utiliser entre parenthèses :

- *fa_top*
- *fa_middle*
- *fa_center*

**b. Dessiner des formes**

Game Maker possède un ensemble de fonctions vous permettant de dessiner différentes formes géométriques. Celles-ci peuvent être très basiques (cercle, carré, segment…) ou plus complexes (flèche, rectangle à coins arrondis, ellipse…). Voici quelques-unes de ces fameuses fonctions :



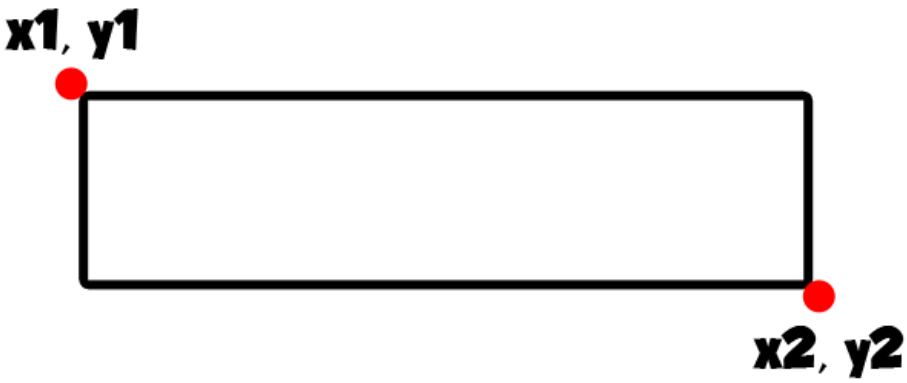
Copyright © creersonjeu.fr

- **draw_rectangle(*x1, y1, x2, y2, contour de la forme*) ;**

  Comme son nom l'indique, cette fonction permet de dessiner un rectangle. Elle existe également dans une forme alternative pour dessiner un rectangle à bords arrondis : **draw_roundrect().**

  Comment dessiner notre rectangle ?

  Les variables *x1* et *y1* correspondent à la position du coin supérieur gauche, tandis que *x2* et *y2* correspondent au coin inférieur droit. La dernière variable peut être égale à *true* ou *false*, et permet de dessiner un rectangle plein (*false*) ou simplement sa bordure (*true*). Si la bordure est dessinée, elle sera d'une largeur de 1 pixel.




- **draw_circle (*x, y , r, contour de la forme*) ;**

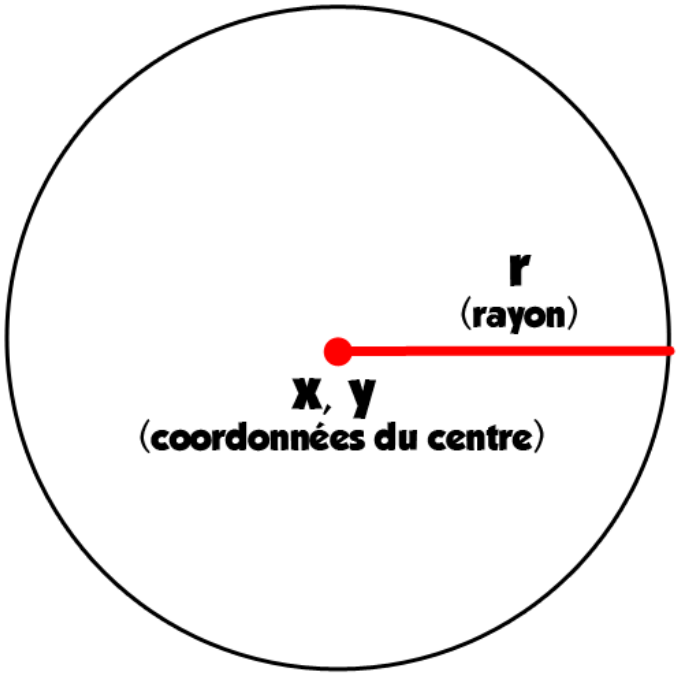
  Cette fonction ci permet de dessiner un cercle. Rien de nouveau par rapport au dessin de rectangle, si ce n'est qu'il faut connaître sa géométrie de base !

  En effet, le *x* et le *y* correspondent aux coordonnées du centre du cercle. Le *r* quant à lui représente son rayon. Pour rappel, le rayon d'un cercle est selon la définition du Larousse :

  Un segment dont une extrémité est le centre d'un cercle, d'une sphère, d'un disque, d'une boule, l'autre étant un point du cercle, de la sphère ou de la frontière du disque ou de la boule.

  Pour ceux pour qui cela serait encore un peu flou, voici une image explicative :


Copyright © creersonjeu.fr




Enfin comme pour le rectangle, la dernière variable peut prendre les valeurs true ou false et servent à dessiner soit un cercle plein (false) soit un contour de cercle (true).

- **draw_line (*x1, y1, x2, y2*) ;**

  Cette fonction permet de dessiner un segment de 1 pixel de largeur. Les variables *x1* et *y1* correspondent à l'extrémité gauche du segment, tandis que *x2* et *y2* correspondent à l'extrémité droite.

Il en existe bien entendu de nombreuses autres, que je vous encourage à aller voir sur la documentation officielle ☺



Copyright © creersonjeu.fr

### c. Bonus : dessiner une barre de vie !

Il existe dans Game Maker une fonction bien pratique qui ravisera tous ceux qui sont en train de créer ou prévoit de créer un RPG. Bon, il n'y a pas de surprise puisque vous avez lu le titre de ce paragraphe donc je l'annonce : oui, une fonction pré-intégrée permet de dessiner une barre de vie, avec sa logique pré-intégrée ! Génial non ?

Nous allons maintenant voir comment celle-ci fonctionne et comment vous pouvez l'incorporer dans votre jeu.

- **Draw_healthbar(x1, y1, x2, y2, amount, backcol, mincol, maxcol, direction, showback, showborder);**

  Vous l'aurez remarqué, j'ai pour habitude de traduire l'ensemble des variables présentes entre les parenthèses des fonctions afin de mieux pouvoir vous les expliquer. Dans ce cas précis, je ne les ai pas écrites afin de nous permettre d'y voir plus clair. Les variables d'origine (en anglais) sont présentées comme sur la documentation officielle.

  Comment fonctionne-t-elle ?

  Vous devez surement reconnaitre les premières variables x1, y1, x2 et y2, puisqu'elles reprennent le principe de la fonction **draw_rectangle() ;**

  Nous allons plutôt nous intéresser aux variables suivantes :

  - *amount* : correspond au montant de points de vie affichés par la barre. Je vous conseille personnellement d'utiliser une variable dédiée, que vous créerez dans l'event *create* de votre objet (voir note ci-dessous)
  - *backcol* : la couleur du fond de la barre (voir *indices des couleurs*).
  - *mincol* : la couleur de la barre lorsque la vie tombe à 0
  - *maxcol* : la couleur de la barre lorsque la vie est au maximum
  - *direction* : la direction de l'affichage de barre en fonction de son point d'origine. Elle peut prendre les valeurs 0 (vers la gauche), 1 (vers la droite), 2 (vers le haut) et 3 (vers le bas)



Copyright © creersonjeu.fr

- *showback* : peut prendre les valeurs *true* ou *false*. Permet de déterminer si le fond de la barre est dessiné ou non
- *showborder* : peut prendre les valeurs *true* ou *false*. Permet de dessiner une bordure de 1 pixel

Note concernant la variable *amount* : la barre de vie peut fonctionner uniquement si cette variable possède une valeur **en pourcentage**. Pour pouvoir afficher la vie de votre personnage par exemple, il faudra donc créer dans son *event create* deux variables. La première étant la vie maximum du personnage (par exemple 100) et la seconde sa vie actuelle. Ceci fait il faudra enfin créer une troisième et dernière variable, qui sera chargée de contenir en pourcentage la vie du personnage. Pour cela il faudra mettre la formule suivante : **(vie du perso/vie max du perso) * 100.**

C'est cette dernière variable que vous renseignerez dans votre fonction. J'imagine que ce n'est pas forcément clair, donc voici un exemple.

Vous pouvez également aller voir mon tutorial *Créer son RPG : Les combats* sur ma chaîne YouTube dans lequel nous avons également utilisé cette fonction pour afficher des barres de vies ! **C'est par ici.**

```
// Points de vie du chevalier

pdv = 100; // Points de vie actuels
max_pdv = 100; // Points de vie maximum

```

```
// Calcul en pourcentage des points de vie du chevalier

pdvp = (chevalier.pdv / chevalier.max_pdv) * 100;

// Ecriture de la healthbar, la variable "pdvp" correspond au pdv en %

draw_healthbar(450, 340, 580, 355, pdvp, c_black, c_red, c_lime, 0, true, true);
```

Résultat :

Copyright © creersonjeu.fr

### d. Les indices de couleurs dans Game Maker

Nous avons abordé le sujet plusieurs fois au cours des derniers paragraphes : les couleurs. Comment dire à Game Maker que vous voulez afficher telle ou telle couleur ? Est-ce qu'il suffit d'écrire « bleu » ?

En réalité non, mais ce n'est pas beaucoup plus difficile que ça. Il existe dans Game Maker différentes constantes, qui correspondent chacune à une couleur différente. Je vous ai fait un petit tableau récapitulatif (que vous pouvez également retrouver dans la documentation officielle) :

| Constante | Couleur |
|---|---|
| *c_aqua* | |
| *c_black* | |
| *c_blue* | |
| *c_fuchsia* | |
| *c_gray* | |
| *c_green* | |
| *c_lime* | |
| *c_ltgray* | |
| *c_maroon* | |
| *c_navy* | |
| *c_olive* | |
| *c_orange* | |



Copyright © creersonjeu.fr

| | |
|---|---|
| *c_purple* | |
| *c_red* | |
| *c_silver* | |
| *c_teal* | |
| *c_white* | |
| *c_yellow* | |

Il existe une fonction que nous n'avons pas encore abordée et qui pourtant vous sera très utile pour assigner des couleurs :

- **draw_set_colour(*couleur à assigner*) ;**

Ecrite en amont de vos fonctions de dessin, elle permet d'assigner une couleur par défaut à tout ce qui sera affiché grâce à ces dernières.

Exemple :

```
1  draw_set_colour(c_blue);
2
3  draw_text(30, 70, "Bonjour !");
4
5  // Le texte sera ainsi dessiner en bleu !
```

Cela permet par exemple d'éviter l'utilisation de la fonction **draw_text_colour().**



Copyright © creersonjeu.fr

# FONCTIONS LIÉES AUX CONTRÔLES IN-GAME

Créer un jeu visuellement impressionnant, avec une histoire élaborée et de nombreuses heures de jeux c'est bien, mais il ne faut pas oublier également l'un des facteurs majeurs qui font qu'un jeu est bon : le gameplay.

Le gameplay se travaille de deux manières différentes :

- au sein du jeu, à travers une mécanique spécifique, avec laquelle le joueur va devoir se familiariser.
- dans le monde réel, à travers les différents moyens de contrôle que vous allez mettre à la disposition du joueur, afin que ce dernier puisse interagir avec votre monde virtuel.

C'est évidemment sur la seconde option que nous allons nous concentrer pour cette partie. Je vais vous présenter ici différentes fonctions qui vous permettront de programmer une interface au clavier, à la souris, à la manette et même tactile pour votre jeu.

### a. Programmer des interactions au clavier

Il existe différentes fonctions qui vous permettront de programmer des contrôles clavier pour votre jeu. Ces dernières seront à utiliser grâce à un ensemble de constantes et fonctions spécifiques déjà incorporées dans Game Maker, qui ont pour but d'identifier chaque touche du clavier.

Nous allons dans un premier temps nous attarder sur la reconnaissance des touches par le logiciel.

Il existe tout d'abord une fonction très pratique, qui vous permettra d'identifier la plupart des touches.



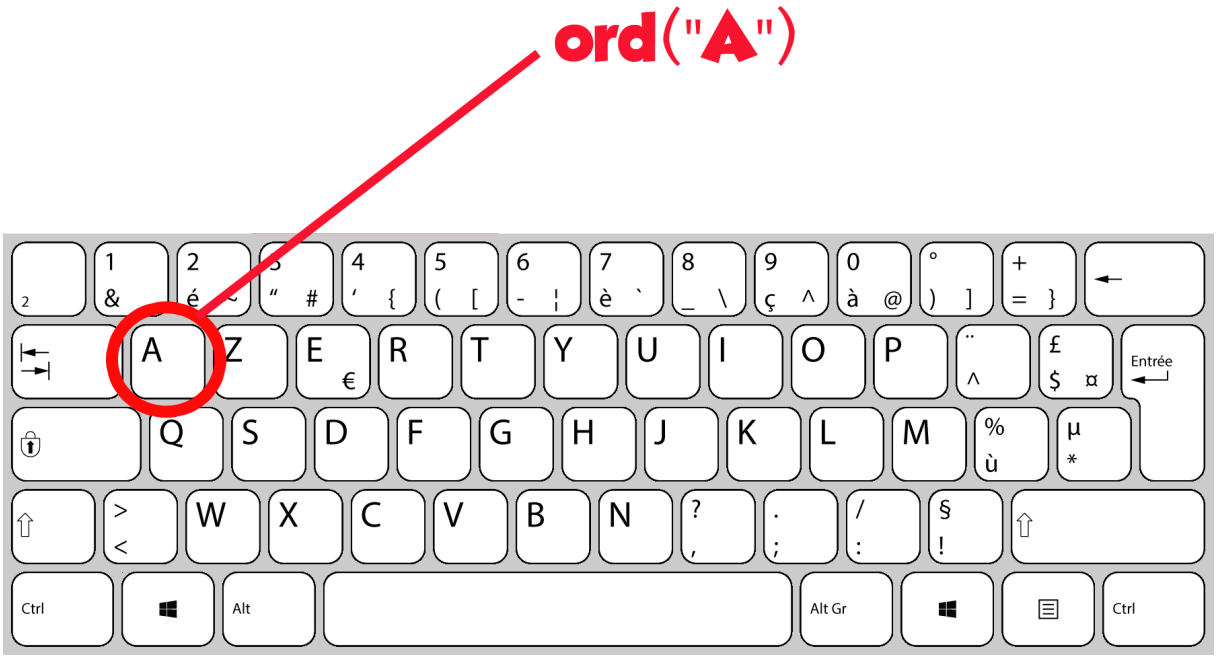
Copyright © creersonjeu.fr

- **ord(*« touche en majuscule »*) ;**

  Cette fonction permet de définir quelle touche doit être testée. Elle fonctionne pour l'ensemble des touches représentant les lettres de l'alphabet ou les chiffres. Son utilisation est très simple, il suffit de renseigner la lettre (ou le chiffre) à tester entre les parenthèses et en majuscule entre guillemets.

  Si l'on veut tester la touche « A » par exemple, il suffira d'écrire **ord(« A ») ;**

  Attention : veillez à toujours mettre la lettre en majuscule ! Sinon cela ne marchera pas.




  Exemple :

  On veut tester si la touche du clavier A est appuyée. Si c'est le cas, alors le jeu s'arrête.

```
if (keyboard_check(ord("A")));
{
    game_end();
};
```



Copyright © creersonjeu.fr

Nous savons désormais comment tester les touches alphabétiques et numériques de notre clavier. Mais comment faire si l'on souhaite tester la touche « entrée », « espace » ou les flèches directionnelles ?

Et bien il existe des constantes pour cela ! Ces dernières commencent toutes par « *vk_* ». Vk signifie « *Virtual Keyboard* ». Voici ci-dessous un petit tableau non exhaustif des différentes constantes intégrées à Game Maker. (Pour la liste complète, je vous encourage comme d'habitude à vous référer à la documentation officielle Game Maker ;)

| Constante | Touche |
|---|---|
| *vk_anykey* | Si une touche du clavier est appuyée |
| *vk_left* | Touche flèche gauche |
| *vk_right* | Touche flèche droite |
| *vk_up* | Touche flèche haut |
| *vk_down* | Touche flèche bas |
| *vk_enter* | Touche entrée |
| *vk_escape* | Touche echap |
| *vk_space* | Touche espace |
| *vk_shift* | Touches shift |
| *vk_alt* | Touches alt |
| *vk_backspace* | Touche backspace |
| *vk_tab* | Touche tab |
| *vk_delete* | Touche suppression |



Copyright © creersonjeu.fr

| | |
|---|---|
| *vk_prinrtscreen* | Touche impression écran |
| *vk_f1 → f12* | Touches de f1 à f12 |
| *vk_numpad0 → numpad9* | Touches du numpad de 0 à 9 |
| *vk_multiply* | Touche multiplier sur le numpad (*) |
| *vk_divide* | Touche diviser sur le numpad (/) |
| *vk_add* | Touche additionner sur le numpad (+) |
| *vk_substract* | Touche soustraire sur le numpad (-) |
| *vk_decimal* | Touche de marquage des décimales sur le numpad (.) |

**b. Tester les interactions avec le clavier**

Lorsque l'on configure les contrôles pour son jeu, il faut pouvoir demander à Game Maker quelles touches ont été appuyées afin de pouvoir programmer un gameplay efficace. Heureusement, il existe plusieurs fonctions permettant de vérifier ces interactions.

- **Keyboard_check(*« touche »*) ;**

  Cette fonction permet de tester si une des touches du clavier est **en train** d'être appuyée. Il vous suffit de mettre entre les parenthèses la fonction **ord() ;** ou l'une des constantes que nous avons vu précédemment. Cette fonction retournera *true* pendant tout le temps ou la touche est appuyée.

  Exemple :


Copyright © creersonjeu.fr

Si la touche flèche droite est appuyée, alors l'objet se déplace de 3 pixels vers la droite (sur l'axe des x).

```
if (keyboard_check(vk_right))
{
    x += 3;
};
```

- **Keyboard_check_pressed(*« touche »*) ;**

  Contrairement à la fonction précédente, celle-ci permet de tester si une **touche a été appuyée**. Cela veut dire qu'elle retournera *true* **uniquement** au moment où la touche est pressée. Afin qu'elle puisse de nouveau être appelée et retourner *true*, il faudra que le joueur relâche la touche et appuie dessus une nouvelle fois.

  Cette fonction peut être très pratique lorsqu'elle est associée à une action qui nécessite seulement un appui de touche. Imaginez par exemple que lorsque l'on appuie sur espace le joueur saute. Si on utilise la fonction précédente **keyboard_check() ;** cette dernière retournera *true* tant que le joueur n'aura pas relâcher la barre d'espace. Son personnage s'envolera dans les airs au lieu de retomber au sol après avoir sauté ! Il faudra donc utiliser **keyboard_check_pressed() ;**

- **Keyboard_check_released(*« touche »*) ;**

  Cette fonction fonctionne de la même manière que la précédente, à la différence qu'elle retournera *true* uniquement lorsque la touche testée aura été appuyée puis relâchée.

### c. Programmer des interactions à la souris

Nous venons de voir comment tester et programmer les interactions au clavier. La suite logique est donc de voir maintenant comment tester les inputs de notre souris !



Copyright © creersonjeu.fr

Tout comme pour le clavier, Game Maker a mis en place un certain nombre de constantes permettant d'identifier les différents boutons d'une souris. Vous pouvez les retrouver dans le tableau ci-dessous :

| Constante | Bouton |
|---|---|
| *mb_left* | Bouton gauche |
| *mb_middle* | Bouton central (roulette) |
| *mb_right* | Bouton droit |
| *mb_none* | Si aucun bouton n'est appuyé |
| *mb_any* | Lorsqu'un des boutons est appuyé |

**d. Tester les interactions avec la souris**

Voyons maintenant les différentes fonctions permettant de tester les boutons de notre souris.

- **Mouse_button ;**

  Petite exception de notre liste, car c'est une variable ! Celle-ci sert à tester quel bouton est actuellement appuyé par le joueur.

  Voici un exemple de son utilisation :

  Nous allons tester si le bouton gauche de la souris est appuyé. Si la condition retourne *true,* alors la variable shoot retourne *true*.

```
1 if (mouse_button = mb_left)
2 {
3     var_shoot = true;
4 };
```



Copyright © creersonjeu.fr

- **Mouse_check_button(constante bouton);**

  Fonctionne de la même manière que la fonction **keyboard_check()** vu précédemment. Il vous suffit de mettre une des constantes présentes dans le tableau du début de chapitre pour tester une touche.

- **Mouse_check_button_pressed(constante bouton);**

  Fonctionne de la même manière que la fonction **keyboard_check_pressed().**

- **Mouse_check_button_released (constante bouton);**

  Fonctionne de la même manière que la fonction **keyboard_check_released ().**

- **Mouse_wheel_up();**

  Cette fonction permet de tester si la roulette de la souris est tournée vers le haut par le joueur. Retourne *true* si c'est le cas, sinon *false*.

- **Mouse_wheel_down();**

  Cette fonction permet de tester si la roulette de la souris est tournée vers le bas par le joueur. Retourne *true* si c'est le cas, sinon *false*.

### e. Programmer des interactions avec un gamepad

Jouer à son jeu au clavier et à la souris peut devenir à la longue ennuyeux. Tous les joueurs vous le diront, pouvoir jouer sa partie avec une manette dans les mains est souvient bien plus confortable et pratique !

Nous allons voir rapidement ici comment programmer les interactions avec les différents boutons d'une manette.

Attention, la compatibilité n'est pas forcément la même avec toutes les marques de manettes de jeu. Je vais vous décrire ici uniquement ce que j'ai déjà testé, c'est-à-dire la programmation pour une manette d'Xbox 360 (c'est d'ailleurs cette dernière qui est décrite



Copyright © creersonjeu.fr

dans la documentation officielle). Pour les autres marques telle que celles de Sony Playstation, je vous laisse faire vos propres tests. N'hésitez pas à m'envoyer un email pour me faire un retour sur la compatibilité !

Sur Windows vous pouvez connecter jusqu'à 12 gamepads (numérotés de 0 à 11), tandis que Mac 4 au maximum sont supportés. Les numéros des gamepads sont appelés par Game Maker des *slots*.

Note : en réalité les compatibilités changent en fonction du système d'exploitation vers lequel est exporté votre jeu. Sur mac par exemple, seules les manettes d'Xbox 360 et Playstation3 sont supportées à l'heure où j'écris ces lignes.

Comme pour les autres périphériques, il existe des constantes préenregistrées que nous allons décrire dans le tableau ci-dessous :

| Constante | Bouton dédié |
|---|---|
| ***gp_face1*** | *Bouton A* |
| ***gp_face2*** | *Bouton B* |
| ***gp_face3*** | *Bouton X* |
| ***gp_face4*** | *Bouton Y* |
| ***gp_shoulderl*** | *Bouton LB* |
| ***gp_shoulderlb*** | *Bouton LT* |
| ***gp_shoulderr*** | *Bouton RB* |
| ***gp_shoulderrb*** | *Bouton RT* |
| ***gp_select*** | *Bouton Back* |
| ***gp_start*** | *Bouton Start* |

| | |
|---|---|
| ***gp_stickl*** | *Bouton stick gauche* |



Copyright © creersonjeu.fr

| | |
|---|---|
| ***gp_stickr*** | *Bouton stick droit (appuyé)* |
| ***gp_padu*** | *DPad Haut* |
| ***gp_padd*** | *DPad Bas* |
| ***gp_padl*** | *DPad Gauche* |
| ***gp_padr*** | *DPad Droite* |
| ***gp_axislh*** | *Axe horizontal du stick gauche (x1)* |
| ***gp_axislv*** | *Axe vertical du stick gauche (y1)* |
| ***gp_axisrh*** | *Axe horizontal du stick droit (x2)* |
| ***gp_axisrv*** | *Axe vertical du stick droit (y2)* |

**f. Tester les interactions avec le gamepad**

- **gamepad_button_check(slot, constante bouton);**
- **gamepad_button_check_pressed(slot, constante bouton);**
- **gamepad_button_check_released(slot, constante bouton);**

J'ai cette fois-ci regroupé les 3 fonctions car je pense qu'il n'est plus nécessaire de les présenter. Ces dernières fonctionnent exactement de la même manière que pour les autres périphériques en retournant *true* en fonction de l'état du bouton testé.

Il vous suffit comme d'habitude de définir entre parenthèses le numéro du gamepad à tester, ainsi que la constante voulue.

Ces fonctions sont parfaites pour tester les boutons de notre gamepad, mais qu'en est-il ? Rassurez-vous, il existe également une fonction dédiée !



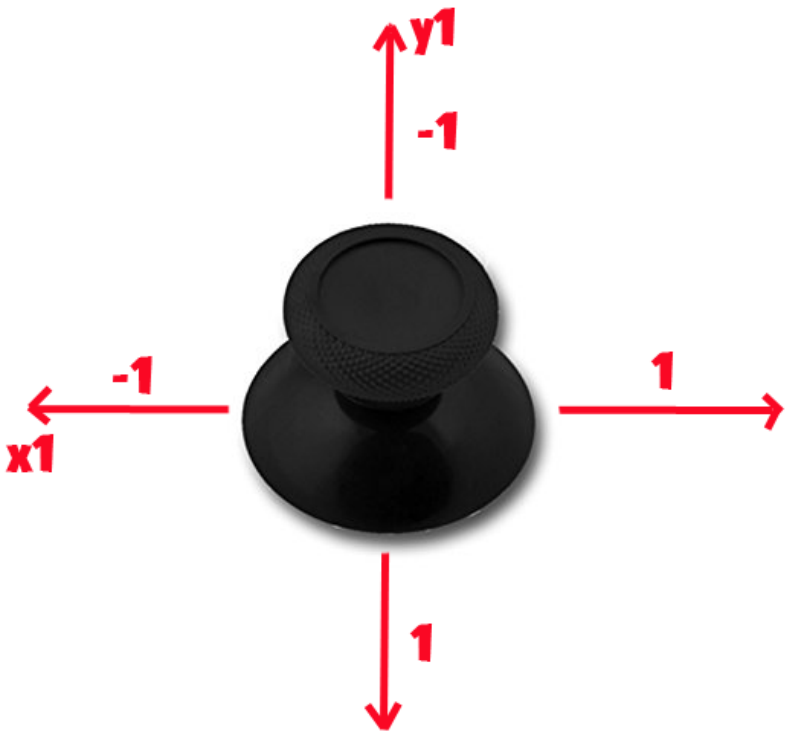
Copyright © creersonjeu.fr

- **gamepad_axis_value(slot, constante stick);**

  Cette fonction va vous permettre de tester l'interaction du joueur avec les sticks du gamepad.

  Comment ça marche ?

  Pour que vous compreniez bien, il faut savoir que Game Maker interprète les mouvements des sticks selon leur axe. Soit horizontal (représenté par les variables x1 pour le stick gauche, x2 pour le stick droit) soit vertical (respectivement y1 et y2 pour les sticks gauche et droit). Ces variables peuvent, en fonction du mouvement du stick, prendre une valeur comprise entre -1 et 1.




  Il existe différentes méthodes pour tester les interactions des sticks et leur attribuer un mouvement dans le jeu. Je vais vous présenter ici la méthode la plus simple, que j'utilise et qui est expliquée quelque part sur le site officiel Game Maker.

  Pour la petite explication, nous allons dans notre objet *Player*, dans l'event *step*, créer deux variables. La première représentera les mouvements horizontaux, tandis que la seconde les mouvements verticaux. Si leur valeur est différente de 0 (c'est-à-


Copyright © creersonjeu.fr

dire que les sticks sont en mouvement) alors on multipliera la valeur retournée par une valeur de vitesse (on pourrait également créer une variable « *speed* » dans l'event *create* de notre objet).



Copyright © creersonjeu.fr

# LE MOT DE LA FIN

Voilà les créateurs, c'est tout pour cet ouvrage. J'espère que vous avez appris des tonnes de nouvelles choses et que ce livre vous servira de manuel de référence pour vos prochaines créations (on ne sait jamais) !

N'hésitez pas à venir me dire sur les réseaux sociaux ce que vous en avez pensé, cela me fera très plaisir ☺

Sur ce je vous dis au revoir, et à très bientôt sur le blog et la chaîne Youtube pour de nouveaux tutoriaux !

Maxime

# MERCI !

*YouTube*

*Twitter*

*Facebook*



Copyright © creersonjeu.fr